병렬가상기계(PVM)서고함수

국가과학원 콤퓨터과학연구소

주체 98(2009)년 4월

병렬가상기계의 서고함수들

여기서는 모든 병렬가상기계함수들을 자모순서로 서술한다. 매 함수들은 C언어와 포트란언어에 대하여 따로 따로 설명한다. 또한 매 함수들에 대한 실례프로그람들도 제시하였다.

pvmfaddhost() pvm_addhosts()

한개이상의 처리기들을 가상기계에 첨부한다.

문법

C:

```
int info = pvm_addhosts( char **hosts, int nhost,
                         int  *infos )
```

포트란:

```
call pvmfaddhost( host, info )
```

변수:

hosts - 추가하여야 할 처리기들의 이름을 포함한 문자배렬에로의 지적자

nhost - 추가해야 할 처리기들의 개수를 지적하는 옹근수

infos - 이 함수가 돌려주는 개별적인 처리기들의 상태코드를 가지고있는 길이가 nhost인 옹근수배렬, 오유가 생긴 경우에는 령보다 작은 값이 들어있다.

host - 추가해야 할 처리기의 이름이 들어있는 문자배렬

info - 이 함수가 되돌리는 상태코드, nhost보다 작은 값은 부분적인 실패를 나타내며 1보다 작으면 모두 실패하였음을 보여준다.

설명:

pvm_addhosts는 가상기계를 구성하기 위하여 현재의 처리기들의 구성에 hosts로 지적한 처리기들의 목록을 첨가한다.

만일 pvm_addhosts가 성공하면 info는 nhost와 같다.

부분적으로 실패하는 경우에는 1 <= info < nhost이며 완전한 실패이면 info < 1이다. 어느 처리기에서 오유가 일어났는가는 배렬

infos를 검사하여 결정한다. 포트란함수 pvmfaddhost는 한번 호출에서 한개의 처리기만을 추가한다. 실패하면 병렬가상기계체계는 이 함수를 다시 호출한다.

리용자는 이 기능을 리용하여 응용프로그람의 오유허용능력을 개선할수 있다. 응용프로그람에서는 pvm_mstat와 pvm_config를 리용하여 처리기들의 상태를 문의할수 있다.

만일 처리기가 실패하면 그 처리기는 자동적으로 처리기구성에서 지워진다. pvm_addhosts지령으로 리용자는 응용프로그람에서 새로운 처리기를 추가할수 있다.

이것은 응용프로그람작성자가 프로그람을 작성할 때 처리기의 실패에 대응하여 오유허용능력을 높일수 있도록 한다.

또한 이 특징을 리용하면 프로그람작성자가 가능한껏 많은 처리기들을 추가할수 있으며 보다 많은 계산능력을 리용할수 있게 한다.

실례 프로그람

C:

```
static char *hosts[] = {
                        "sparky",
                        "thud.cs.utk.edu"
                        };
info = pvm_addhosts( hosts, 2, infos );
```

포트란:

```
CALL PVMFADDHOST( 'azure', INFO )
```

오유

pvm_addhosts가 돌려주는 오유코드는 아래와 같다.

이름	가능한 경우
PvmBadParam	틀린변수값을 지정
PvmAlready	이미 추가되였다.
PvmSysErr	국부 pvmd가 응답하지 않는다.

infos의 오유코드는 다음과 같다.

이름	가능한 경우
PvmBadParam	처리기이름이 틀렸다.
PvmNoHost	그러한 처리기가 없다.
PvmCantStart	처리기에서 pvmd시동이 실패
PvmDupHost	처리기는 이미 구성되여있다.
PvmBadVersion	원격 pvmd의 판본이 일치하지 않는다.
PvmOutOfRes	PVM은 체계자원밖에서 실행한다.

pvmfbarrier() pvm_barrier()

그룹에 있는 모든 프로세스들이 자기를 호출할 때까지 호출한 프로세스를 봉쇄한다.

문법

C:

```
int info = pvm_barrier( char *group, int count )
```

포트란:

```
call pvmfbarrier( group, count, info )
```

변수:

group – 문자렬로 된 그룹이름, 그룹은 존재해야 하며 호출한 프로세스는 이 그룹성원이여야 한다.

count – 그룹성원들을 모두 해방하기전에 pvm_barrier를 호출하는 그룹성원들의 개수를 지적하는 옹근수, count는 대체로 지적된 그룹의 전체 성원들의 개수를 표시한다.

info – 이 함수가 되돌리는 상태코드, 오유가 발생하면 령보다 작다.

설명

pvm_barrier는 그룹에 있는 count개수의 성원들이 모두 pvm_barrier를 호출할 때까지 호출한 프로세스를 봉쇄한다. 다른 프로세스들이 pvm_barrier를 호출한 다음에 프로세스들이 주어진 그룹에 결합할수 있기때문에 count변수가 요구된다.

따라서 병렬가상기계는 몇개의 그룹성원들이 기다리고있는가를 알지 못한다. count는 좀 작게 설정할수 있어도 일반적으로는 전체 그룹성원들의 개수를 지적한다. 따라서 pvm_barrier호출의 론리적인 기능은 그룹동기화를 지원하는것이다.

어떤 시점에서 장벽연산에 협력하는 모든 성원들은 같은 count값으로 이 함수를 호출해야 한다. 일단 주어진 장벽연산이 성공하면 같은 그룹이름을 가지고 같은 그룹에서 다시 호출할수 있다. 특수한 경우에 count 가 1이면 병렬가상기계는 pvm_gsize()의 값을 리용한다(즉 모든 그룹성원들). 이 경우는 그룹들을 창조하고 실행기간에는 그 그룹이 변하지 않는 경우에 쓸모가 있다. pvm_barrier는 성공하면 info는 0으로 된다. 오유가 생기면 info < 0으로 된다.

실례프로그람

C:

```
        inum = pvm_joingroup( "worker" );
                    .
                    .
        info = pvm_barrier( "worker", 5 );
```

포트란:

```
      CALL PVMFJOINGROUP( "shakers", INUM )
      COUNT = 10
      CALL PVMFBARRIER( "shakers", COUNT,
                                   INFO )
```

오유

pvm_barrier가 돌려주는 오유들은 다음과 같다.

이름	가능한 경우
PvmSysErr	pvmd가 시동하지 않았거나 파괴되였다.
PvmBadParam	count < 1이다.
PvmNoGroup	존재하지 않는 그룹이름을 주었다.
PvmNotInGroup	호출프로쎄스는 주어진 그룹성원이 아니다.

능동통보완충기에 있는 자료를 방송한다.

문법

C:

```
int info = pvm_bcast( char *group, int msgtag )
```

포트란:

```
call pvmfbcast( group, msgtag, info )
```

변수:

group – 현존 그룹의 그룹이름

msgtag – 리용자가 준 옹근수 통보표적, msgtag >= 0이여야 한다. 이것은 리용자프로그람이 여러가지 종류의 통보들을 구별할수 있게 한다.

info – 이 함수가 돌려주는 옹근수 상태코드, 오유인 경우 귀환값은 0보다 작다.

설명

pvm_bcast는 능동통보완충기에 있는 모든 통보들을 그룹에 있는 모든 성원들에게 방송한다. PVM 3.2에서 방송통보는 송신자에게 돌아오지 않는다. 임의의 병렬가상기계과제도 pvm_bcast()를 호출할수 있으며 그룹의 성원이 아니라도 된다. 통보의 내용은 msgtag에 따라 구별된다. pvm_bcast가 성공하면 info > 0이며 오유가 발생하면 info < 0으로 된다. pvm_bcast는 비동기함수이다.

송신측에서 연산은 통보가 수신프로쎄스에 안전하게 도착한다면 즉시 회복된다. 이 절차는 수신측에서 정확히 자료를 접수할 때까지 송신측에서의 연산이 중지되는 동기식통신과는 완전히 대조적이다.

pvm_bcast는 먼저 그룹자료기지들을 검사하여 그룹성원들의 tid들을 결정한다. 이 tid들에 대하여 다중통신이 진행된다. 만일 그룹이 방송통신기간에 변경된다고 해도 이 변경은 방송통신에 영향을 주지 않는다.

실례프로그람

C:

```
      info = pvm_initsend( PvmDataRaw );
      info = pvm_pkint( array, 10, 1 );
      msgtag = 5;
      info = pvm_bcast( "worker", msgtag );
```

포트란:

```
      CALL PVMFINITSEND( PVMDEFAULT )
      CALL PVMFPKFLOAT( DATA, 100, 1, INFO )
      CALL PVMFBCAST( 'worker', 5, INFO )
```

오유

pvm_bcast는 다음의 오유를 되돌린다.

이름	가능한 경우
PvmSysErr	pvmd가 기동하지 않았거나 파괴되였다.
PvmBadParam	허용되지 않은 msgtag
PvmNoGroup	존재하지 않는 그룹이름

pvmfbufinfo()　　　　pvm_bufinfo()

요구한 통보완충기의 정보를 돌려준다.

문법

C :

```
   int info = pvm_bufinfo( int bufid, int *bytes,
                           int *msgtag, int *tid )
```

포트란:

```
   call pvmfbufinfo( bufid, bytes, msgtag, tid, info )
```

변수

bufid – 구체적인 통보완충기식별자를 지적하는 옹근수

bytes – 전체 통보에 대하여 바이트단위로 길이를 돌려주는 옹근수

msgtag – 통보표적을 돌려주는 옹근수

tid – 통보의 원천지를 가리키는 옹근수
info – 함수가 돌려주는 옹근수 상태코드, 령보다 작은 값은 오유를 표시한다.

설명

pvm_bufinfo는 요구한 통보완충기의 정보를 돌려준다. 이 함수는 제일 마지막에 접수한 통보의 크기나 원천을 알려준다. pvm_bufinfo는 응용프로그람이 오는 통보를 접수할수 있을 때 특히 쓸모가 있다. 이때 그의 원천 tid나 제일 처음에 도착한 통보와 련관되여있는 msgtag에 따라 다음 동작이 진행된다. pvm_bufinfo가 정상으로 완료하면 info는 0으로 된다. 만일 오유가 발생하면 info < 0일것이다.

실례프로그람

C:

```
bufid = pvm_recv( -1, -1 );
info = pvm_bufinfo( bufid, &bytes, &type, &source);
```

포트란:

```
CALL PVMFRECV( -1, -1, BUFID )
CALL PVMFBUFINFO( BUFID, BYTES, TYPE,
                  SOURCE, INFO )
```

오유들

pvm_bufinfo가 돌려주는 오유코드들은 다음과 같다.

이름	가능한 경우
PvmNoSuchBuf	지적된 완충기가 존재하지 않는다.
PvmBadParam	틀린 변수

pvmfcatchout() pvm_catchout()

자식과제들의 출구를 입력한다.

문법
C:

```
#include <stdio.h>
int bufid = pvm_catchout( FILE *ff )
```

포트란:

```
call pvmfcatchout( onoff )
```

변수
ff – 수집한 출구자료들을 써넣을 파일서술자
onoff – 옹근수변수, 출구집합을 on 혹은 off 하겠는가를 지적한다.

설명
pvm_catchout는 이 함수를 호출한 과제가 이후부터 새롭게 창조한 자식과제들의 표준출구를 받게 한다. 자식과제들이 표준출구나 표준오유출구에 보낸 문자들은 pvmd들이 수집하고 다음에 이것을 조종통보로 어미과제에 보내준다. 자식과제가 PvmOutputTid를 재설정하지 않는다면 손자과제들이 내보내는 출구도 수집한다.
출구의 매 행은 아래와 같은 형식을 가진다.

[txxxxx] BEGIN
[txxxxx] (자식과제로부터 본문)
[txxxxx] END

매 과제의 출구는 한개의 BEGIN행과 한개의 END행을 포함하며 그 사이에 과제의 출구내용을 포함하고있다. C에서는 출구파일서술자를 지적할수 있다. 포트란에서는 on 혹은 off로 출구집합을 바꿀수 있으며 결과는 어미과제의 표준출구에 등록된다. 출구집합이 유효할 때 pvm_exit를 호출하면 그들모두의 출구를 인쇄할수 있도록 통보를 보내는 과제들이 완료할 때까지 봉쇄된다. 이것을 없애자면 프로그람 작성자는 pvm_exit를 호출하기전에 pvm_catchout(0)을 호출하여 출구집합을 off로 하여야 한다. pvm_catchout(1)는 항상 PvmOk를 돌려준다.

실례프로그람

C:

```
        #include <stdio.h>
        pvm_catchout(stdout);
```

포트란:

```
        CALL PVMFCATCHOUT( 1 )
```

오유

pvm_catchout는 오유를 돌려주지 않는다.

pvmfconfig() pvm_config()

현재 가상기계구성에 대한 정보를 돌려준다.

문법

C:

```
   int info = pvm_config( int *nhost, int *narch,
                          struct pvmhostinfo **hostp )
   struct pvmhostinfo {
                          int hi_tid;
                          char *hi_name;
                          char *hi_arch;
                          int hi_speed;
                         } hostp;
```

포트란:

```
        call pvmfconfig( nhost, narch, dtid,
                         name, arch, speed, info )
```

변수

nhost – 가상기계에 있는 처리기들의 개수를 돌려주는 옹근수

narch – 몇개의 자료변환형식이 리용되는가를 지적하는 옹근수

hostp – pvmd의 과제id, 이름, 구성방식, 상대속도를 포함하여 매 처리기들에 대한 정보를 가지고있는 구조체배렬에로의 지적자

dtid – 처리기에 있는 pvmd의 과제id를 돌려준다.

name – 처리기의 이름문자렬
arch – 처리기의 구성방식이름을 돌려주는 문자렬
speed – 처리기의 상대속도를 돌려주는 옹근수, 기정값은 1 000
info – 귀환값, 령보다 작으면 오유를 표시한다.

설명

pvm_config는 현재 가상기계에 대한 정보를 돌려준다. 정보는 조작탁지령 conf으로 얻는 값과 같다. C함수는 한번의 호출로 전체 가상기계에 대한 정보를 얻는다. 포트란함수로는 한번의 호출로 한개의 처리기에 대한 정보를 얻으며 모든 처리기들에 대한 정보를 얻자면 처리기의 개수만큼 함수를 반복호출해야 한다.

pvm_config가 성과적으로 완료하면 info = 0으로 되며 오유가 발생하면 info < 0이다.

실례프로그람

C:

```
info = pvm_config( &nhost, &narch, &hostp );
```

포트란:

```
Do i=1, NHOST
CALL PVMFCONFIG( NHOST, NARCH, DT ID(I),
    HOST(i), ARCH(i), SPEED(i), INFO )
Enddo
```

오유

pvm_config는 다음과 같은 오유를 돌려준다.

이름	가능한 경우
PvmSysErr	pvmd가 응답하지 않는다.

pvmfdelhost() pvm_delhosts()

가상기계로부터 한개 혹은 몇개의 처리기들을 삭제한다.

문법

C:

```
int info = pvm_delhosts( char **hosts, int
                          nhost, int *infos )
```

포트란:

```
call pvmfdelhost( host, info )
```

변수

hosts –삭제해야 할 처리기들의 이름들을 포함하고있는 문자렬들에로의 지적자배렬

nhost – 삭제해야 할 처리기들의 개수를 지적하는 옹근수

infos – 이 함수가 돌려주는 개별적인 처리기들에 대한 상태코드를 포함하고있는 길이가 nhost인 배렬, 오유가 발생하면 령보다 작은 값이 설정된다.

host – 삭제해야 할 처리기의 이름을 포함하고있는 문자렬

info – 귀환 상태코드, nhost보다 작으면 부분오유, 1보다 작으면 완전오유를 표시한다.

설명

pvm_delhosts는 가상기계에서 하나이상의 처리기들을 삭제한다. 이때 이 처리기들에서 실행하고있던 모든 병렬가상기계프로쎄스들과 pvmd는 처리기가 삭제될 때 제거된다.

pvm_delhosts가 성과적으로 완료되면 info=nhost로 될것이다.

1 <= info < nhost 이면 부분적으로 실패하였음을 보여주며 info < 1 이면 완전한 오유가 발생하였다는것을 표시한다.

배렬 infos를 검사하여 어느 처리기에서 오유가 일어났는가를 검사할수 있다. 포트란함수 pvmfdelhost에서는 한번의 호출로 한개의 처리기만을 삭제한다.

어느한 처리기에서 오유가 발생하여도 병렬가상기계체계는 계속 기능을 수행하며 이 처리기를 자동적으로 가상기계에서 삭제한다.

응용프로그람에서는 이때 pvm_notify를 호출하여 어느 처리기에서 오유가 생겼는가를 알수 있다.

이것은 프로그람개발자들이 프로그람의 실행중에 발생하는 오유에 대하여 오유처리를 할수 있게 한다.

실례프로그람

C:

```
      static char *hosts[] = {
                              "sparky",
                              "thud.cs.utk.edu"
                               };
      info = pvm_delhosts( hosts, 2 );
```

포트란:

```
      CALL PVMFDELHOST( 'azure', INFO )
```

오유:

이름	가능한 조건
PvmBadParam	변수값이 틀렸다.
PvmSysErr	국부 pvmd가 응답하지 않는다.
PvmOutOfRes	PVM은 체계의 자원밖에 있다.

pvmfexit()   pvm_exit()

이 프로세스를 병렬가상기계에서 삭제할것을 국부 pvmd에 통보한다.

문법

C:

```
        int info = pvm_exit( void )
```

포트란:

```
        call pvmfexit( info )
```

변수

info – 귀환 상태값, 오유가 발생하면 값은 0보다 작다.

설명

pvm_exit는 프로세스를 병렬가상기계에서 삭제할것을 국부 pvmd에 알려준다. 모든 병렬가상기계프로세스들은 중지하거나 안전하게 완료하려고 하는 경우 pvm_exit를 호출할수 있다.

pvm_spawn으로 시동되지 않은 프로세스들도 이 함수를 호출할수 있다.

실례프로그람

C:

```
/* Program done */
  pvm_exit();
  exit();
```

포트란:

```
CALL PVMFEXIT(INFO)
STOP
```

오유:

이름	가능한 경우
PvmSysErr	pvmd가 응답하지 않는다.

pvmffreebuf() pvm_freebuf()

통보완충기를 해방한다.

문법

C:

```
int info = pvm_freebuf( int bufid )
```

포트란:

```
call pvmffreebuf( bufid, info )
```

변수

bufid – 통보완충기식별자

info – 귀환 상태코드, 오유가 발생하면 령보다 작은 값이 반환된다.

설명

pvm_freebuf는 bufid로 지적된 통보완충기와 관련된 기억기를 해방한다. 통보완충기들은 pvm_mkbuf, pvm_initsend 그리고 pvm_recv에 의해서 창조된다.

pvm_freebuf는 성공하면 info =0으로 되며 오유가 발생하면 info < 0으로 된다. 통보를 보냈거나 또 통보완충기가 더이상 필요없을 때 pvm_mkbuf를 호출하여 창조한 통보완충기들을 해방하는데 pvm_freebuf를 리용한다.

수신완충기들은 그것들이 다중완충기로서 리용되지 않는 한 일반적으로는 해방하지 말아야 한다. 그러나 수신완충기들을 해방하는데도 pvm_freebuf를 리용할수 있다.

그러므로 도착은 하였지만 어떤 요인으로 더 이상 필요없는 통보들은 완충기공간을 소비하지 않도록 없애버릴수 있다.

일반적으로 응용프로그람들에서는 다중송수신완충기가 필요없으며 간단히 pvm_initsend함수를 리용하여 기정의 송신완충기를 재설정할수 있다.

다중완충기들이 쓸모있는 경우도 있다. 실례로 병렬가상기계를 리용하는 서고 혹은 시각적대면부들에서와 실행하는 병렬가상기계 응용프로그람들과 호상작용은 하지만 응용프로그람자체의 통신과는 간섭하려고 하지 않는 경우에 다중완충기가 효과적이다.

실례프로그람

C:

```
        bufid = pvm_mkbuf( PvmDataDefault );
                 :
        info = pvm_freebuf( bufid );
```

포트란:

```
        CALL PVMFMKBUF( PVMDEFAULT, BUFID )
                 :
        CALL PVMFFREEBUF( BUFID, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	변수이름이 틀렸다.
PvmNoSuchBuf	주어진 완충기가 무효하다.

pvmfgetinst() pvm_getinst()

병렬가상기계프로쎄스그룹에 있는 실체들의 개수를 돌려준다.

문법

C:

```
int inum = pvm_getinst( char *group, int tid )
```

포트란:

```
call pvmfgetinst( group, tid, inum )
```

변수

group – 현존 그룹의 그룹이름

tid – 병렬가상기계프로쎄스의 옹근수과제식별자

inum – 함수가 돌려주는 실체번호, 실체번호들은 0부터 시작하여 증가한다, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_getinst는 그룹이름 group와 병렬가상기계의 과제식별자를 주며 입구에 대응하는 유일한 실체번호를 돌려준다. pvm_getinst가 성공하면 inum>= 0으로 되며 오유가 생기면 inum < 0이다.

실례프로그람

C:

```
inum = pvm_getinst( "worker", pvm_mytid() );
inum = pvm_getinst( "worker", tid[i] );
```

포트란:

```
CALL PVMFGETINST( 'GROUP3', TID, INU M)
```

오유:

이름	가능한 경우
PvmSysErr	pvmd가 시동되지 않았거나 파괴되였다.
PvmBadParam	무효한 tid값
PvmNoGroup	존재하지 않는 그룹이름
PvmNotInGroup	tid가 속하지 않은 그룹을 지적하였다.

여러가지 libpvm추가선택들을 보여준다.

문법
C:

```
int val = pvm_getopt( int what )
```

포트란:

```
call pvmfgetrbuf( what, val )
```

변수
what – 무엇을 얻겠는가를 정의하는 옹근수, 추가선택들은 다음과 같다.

추가선택	값	의미
PvmRoute	1	경로규칙
PvmDebugMask	2	오유수정 마스크
PvmAutoErr	3	자동오유보고
PvmOutputTid	4	자식과제의 표준출구
PvmOutputCode	5	출구통보표적
PvmTraceTid	6	자식과제들의 추적수단
PvmTraceCode	7	추적통보표적
PvmFragSize	8	통보의 토막크기
PvmResvTids	9	통보들이 예약된 표적들과 TID들에 허용

val – 추가선택값을 지정하는 옹근수, 미리 정의된 값들은 아래와 같다.

추가선택	값	의미
PvmDontRoute	1	련결요구하지 않거나 허가하지 않음
PvmAllowDirect	2	련결요구하지 않고 허가함
PvmRouteDirect	3	련결요구하고 허가함

설명

pvm_getopt를 리용하여 리용자는 병렬가상기계에 설정된 추가선택값들을 볼수 있다.

실례프로그람

C:

```
route_method = pvm_getopt( PvmRoute );
```

포트란:

```
CALL PVMFGETOPT( PVMAUTOERR, VAL )
```

오유

이름	가능한 경우
PvmBadParam	변수가 틀렸다.

pvmfgetrbuf() pvm_getrbuf()

능동수신완충기에 대한 통보완충기식별자를 돌려준다.

문법

C:

```
int bufid = pvm_getrbuf( void )
```

포트란:

```
call pvmfgetrbuf( bufid )
```

변수

bufid – 능동수신완충기에 대한 통보완충기식별자

설명

pvm_getrbuf는 능동수신완충기에 대한 통보식별자 bufid를 돌려주든가 현재 완충기가 없는 경우에는 0을 돌려준다.

실례프로그람

C:

```
bufid = pvm_getrbuf();
```

포트란:

```
CALL PVMFGETRBUF( BUFID )
```

오유

오유 없음

pvmfgetsbuf() pvm_getsbuf()

능동수신완충기에 대한 통보완충기식별자를 돌려준다.

문법

C:

```
int bufid = pvm_getsbuf( void )
```

포트란:

```
call pvmfgetsbuf( bufid )
```

변수

bufid – 능동송신완충기에 대한 통보완충기식별자

설명

pvm_getsbuf는 능동송신완충기에 대한 통보식별자 bufid를 돌려주든가 현재 완충기가 없는 경우에는 0을 돌려준다.

실례프로그람

C:

```
bufid = pvm_getsbuf();
```

포트란:

```
CALL PVMFGETSBUF( BUFID )
```

오유

오유 없음

pvmfgettid() pvm_gettid()

그룹이름과 실체번호로 지적된 프로쎄스의 tid를 돌려준다.

문법

C:

int tid = pvm_gettid(char *group, int inum)

포트란:

call pvmfgettid(group, inum, tid)

변수

group – 현존 그룹의 이름을 가지고있는 문자렬

inum – 그룹에 있는 프로쎄스의 실체개수

tid – 귀환된 옹근수식별자

설명

pvm_gettid는 그룹이름 group와 실체번호 inum으로 지적된 프로쎄스의 tid를 돌려준다. pvm_gettid는 성공하면 tid > 0, 오유가 생기면 tid< 0이다.

실례프로그람

C:

tid = pvm_gettid("worker",0);

포트란:

CALL PVMFGETTID('worker', 5, TID)

오유:

이름	가능한 경우
PvmSysErr	국부 pvmd에 접촉할수 없다.
PvmBadParam	변수가 틀렸다.
PvmNoGroup	그러한 그룹이 존재하지 않는다.
PvmNoInst	그룹에 그러한 실체가 없다.

pvmfgsize() pvm_gsize()

그룹에 현재 있는 성원들의 개수를 돌려준다.

문법

C:

```
int size = pvm_gsize( char *group )
```
포트란:
```
call pvmfgsize( group, size )
```
변수

group – 현존 그룹의 그룹이름문자렬

size – 현재 그룹에 있는 성원들의 개수, 오유가 생기면 0보다 작은 값을 돌려준다.

설명

pvm_gsize는 group라는 이름을 가진 그룹의 크기를 돌려준다. 오유가 생기면 size < 0 으로 된다.

병렬가상기계체계에서 그룹들은 동적으로 변하므로 이 함수는 주어진 그룹의 현재 크기만을 돌려준다.

실례프로그람

C:
```
size = pvm_gsize( "worker" );
```
포트란:
```
CALL PVMFGSIZE( 'group2', SIZE )
```
오유

이름	가능한 경우
PvmSysErr	pvmd가 시동하지 않았거나 파괴되였다.
PvmBadParam	그룹이름이 틀렸다.

pvmfhalt pvm_halt()

병렬가상기계체계 전체를 중지한다.

문법

C:
```
int info = pvm_halt( void )
```
포트란:
```
call pvmfhalt( info )
```

변수
info – 오유상태를 돌려주는 옹근수

설명
pvm_halt는 원격과제들, 원격pvmd, 국부과제들(이 명령을 호출한 과제도 포함), 국부 pvmd를 포함한 병렬가상기계체계 전체를 중지한다.

오유:

이름	가능한 경우
PvmSysErr	국부 pvmd가 응답하지 않는다.

pvmfhostsync()　　　　　　　　　　pvm_hostsync()

병렬가상기계체계로부터 날자, 시간을 얻는다.

문법
C:

```
#include <sys/time.h>
int info = pvm_hostsync( int host, struct timeval
                    *clk, struct timeval *delta )
```

포트란:

```
call pvmfhostsync( host, clksec, clkusec,
                   deltasec, deltausec, info )
```

변수
host – 처리기의 TID
clk, clksec, clkusec – 처리기로부터 표준시간을 돌려준다.
Delta, deltasec, deltausec – 국부시계와 원격시계와의 편차값을 돌려준다.
설명
pvm_hostsync()는 가상기계에 있는 처리기의 시계를 표준화하고 원격시계와 국부시계와의 편차를 돌려준다.

통보전송으로 인한 델타오유를 줄이기 위하여 원격시계를 읽기전과 읽은 후에 국부시계의 표본들을 얻는다. 델타시간은 부의 값을 가진다. 마이크로초마당은 항상 0~999 999까지의 값을 가지며 초마당의 부호는 델타의 부호를 가진다.

C에서 clk나 delta가 빈 지적자로 입력되면 이 변수는 귀환하지 않는다.

오유:

이름	가능한 경우
PvmSysErr	국부 pvmd가 응답하지 않는다.
PvmNoHost	그런 처리기가 없다.
PvmHostFail	처리기에 접근할수 없다.

pvmfinitsend() pvm_initsend()

기정의 송신완충기를 지우고 통보부호화를 지정한다.

문법

C:

```
int bufid = pvm_initsend( int encoding )
```

포트란:

```
call pvmfinitsend( encoding, bufid )
```

변수

encoding – 다음 통보의 부호화수법을 지적하는 옹근수

C에는 다음과 같은 추가선택들이 있다.

부호화	값	의미
PvmDataDefault	0	XDR
PvmDataRaw	1	부호화를 하지 않는다.
PvmDataInPlace	2	자료를 왼쪽에 배치한다.

bufid – 통보완충기식별자, 오유가 발생한 경우 0보다 작은 값이 설정된다.

설명

pvm_initsend는 송신완충기를 지우고 새로운 통보를 압축하기 위한 준비를 한다. 이 자료압축에서 리용되는 부호화수법은 encoding에 의하여 설정된다.

XDR부호화는 기정으로 리용된다. 그 리유는 병렬가상기계는 이 통보를 보내기전에 리용자가 이종의 처리기를 새롭게 첨부하였는지 모르기때문이다.

만일 리용자가 통보를 보내는 처리기의 자료변환형식이 목적처리기의 자료변환형식과 같다는것을 알고있다면 PvmDataRaw부호화를 리용하여 부호화비용을 줄일수 있다.

PvmDataInPlace부호화는 압축할 때에 자료가 왼쪽에 배치 된다는것을 지적한다. 통보완충기는 보낼 항목들에 대한 지적자와 크기만을 가지고있다.

pvm_send를 호출하면 항목들은 리용자의 기억기밖으로 직접 복사된다. 이 추가선택은 통보를 복사하는 회수를 줄일수 있다.

PvmDataInPlace추가선택은 PVM 3.2에서는 리용할수 없다.

pvm_initsend가 성공적으로 완료하면 bufid는 통보완충기의 식별자를 포함한다. 오유가 일어나면 bufid < 0으로 된다.

실례프로그람

C:

```
bufid = pvm_initsend( PvmDataDefault );
info = pvm_pkint( array, 10, 1 );
msgtag = 3 ;
info = pvm_send( tid, msgtag );
```

포트란:

```
CALL PVMFINITSEND(PVMRAW, BUFID)
CALL PVMFPACK( REAL4, DATA, 100, 1, INFO )
CALL PVMFSEND( TID, 3, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	무효한 부호화

PvmNoMem	malloc가 실패, 완충기를 창조할수 있는 기억기가 부족

pvmfjoingroup() pvm_joingroup()

주어진 그룹에 호출한 프로쎄스를 기입한다.

문법

C:

```
int inum = pvm_joingroup( char *group )
```

포트란:

```
call pvmfjoingroup( group, inum )
```

변수

group – 현존 그룹의 그룹이름

inum – 이 함수가 되돌리는 옹근수실체번호, 실체번호들은 0부터 시작하여 증가한다. 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_joingroup는 group라는 이름을 가진 그룹에 호출한 과제를 등록하며 그룹에서 이 과제의 실체번호 inum을 돌려준다.

만일 오유가 생기면 inum <0으로 된다. 그룹들을 리용할 때 (group, inum) 쌍은 병렬가상기계프로쎄스들을 유일하게 식별한다. 이것은 이전의 병렬가상기계이름달기수법과 일치한다. 만일 과제가 pvm_lvgroup을 호출하여 이 그룹을 떠났다가 같은 그룹에 다시 결합하면 이때 과제가 같은 실체번호를 가지게 된다는것을 담보하지 않는다.

병렬가상기계는 낡은 실체번호들을 다시 리용하려고 하므로 과제가 그룹에 결합할 때 과제는 가능한 실체번호들중에서 제일 작은 값을 가지게 된다.

병렬가상기계에서 과제는 동시에 여러 그룹들의 성원으로 될수 있다.

실례프로그람

C:

```
inum = pvm_joingroup( "worker" );
```

포트란:

```
CALL PVMFJOINGROUP( 'group2', INUM )
```

오유:

이름	가능한 경우
PvmSysErr	pvmd는 시동하지 않았거나 파괴되였다.
PvmBadParam	빈 그룹이름이 지적되였다.
PvmDupGroup	이미 그룹에 결합되여있다.

pvmfkill() pvm_kill()

지적된 병렬가상기계프로쎄스들을 중지한다.

문법

C:

```
int info = pvm_kill( int tid )
```

포트란:

```
call pvmfkill( tid, info )
```

변수

tid – 중지해야 할 병렬가상기계프로쎄스의 과제식별자

info – 귀환된 상태코드, 오유가 발생하면 0보다 작은 값이 설정된다.

설명

pvm_kill은 tid로 지적된 병렬가상기계프로쎄스에 중지신호(SIGTERM)을 보낸다. 다중처리기들인 경우에는 처리기들을 중지하기 위하여 처리기에 의존하는 방법들로 교체된다.

pvm_kill이 성과적으로 완료하면 info = 0으로 되며 오유가 발생하면 info < 0으로 된다. pvm_kill은 호출한 프로쎄스는 중지하지 않는다.

자기자신을 중지하려면 C에서는 pvm_exit()를 호출하고 다음에 exit()를 호출한다. 포트란에서는 pvmfexit를 호출하고 다음에 stop를 호출한다.

실례프로그람

C:

```
info = pvm_kill( tid );
```

포트란:

```
CALL PVMFKILL( TID, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	틀린 tid값을 주었다.
PvmSysErr	pvmd가 응답하지 않는다.

pvmflvgroup() pvm_lvgroup()

호출한 프로세스를 그룹에서 삭제한다.

문법

C:

```
int info = pvm_lvgroup( char *group )
```

포트란:

```
call pvmflvgroup( group, info )
```

변수:

group – 현존 그룹의 그룹이름

info – 상태코드, 오유가 발생하면 0보다 작은 값이 설정된다.

설명

pvm_lvgroup은 그룹 group로부터 호출한 프로세스를 삭제한다. 오유가 발생하면 info <0으로 된다.

만일 프로세스가 pvm_lvgroup나 pvm_exit를 호출하여 그룹에서 떠났다가 후에 다시 같은 그룹에 결합하면 새로운 실체번호가 그 프로세스에 분배된다.

낡은 실체번호들은 pvm_joingroup를 호출한 프로세스에 다시 분배된다.

실례프로그람

C:

```
info = pvm_lvgroup( "worker" );
```
포트란:
```
CALL PVMFLVGROUP( 'group2', INFO )
```

오유:

이름	가능한 경우
PvmSysErr	pvmd가 응답하지 않는다.
PvmBadParam	빈 그룹이름을 지적하였다.
PvmNoGroup	존재하지 않는 그룹이름을 지적하였다.
PvmNotInGroup	이 프로쎄스는 이미 그룹에서 삭제되였다.

pvmfmcast() pvm_mcast()

능동통보완충기에 있는 자료를 과제들의 모임에 다중통신한다.

문법

C:
```
int info = pvm_mcast( int *tids, int ntask, int
                      msgtag )
```
포트란:
```
call pvmfmcast( ntask, tids, msgtag, info )
```
변수

ntask – 통보를 보내야 할 과제들의 개수를 지적하는 옹근수

tids – 통보를 보내야 할 과제들의 과제ID들을 포함하고있는 적어도 ntask길이를 가진 옹근수배렬

msgtag – 리용자가 제공한 통보표적, msgtag>= 0이여야 한다.

info – 상태코드, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_mcast는 능동송신완충기에 보관된 통보들을 tids배렬에 지적된 ntask개의 과제들에 다중통신한다.

이 통보는 호출자의 tid가 tids배렬에 있다 해도 호출자에게는 보내지 않는다. 통보의 내용은 msgtag로 구분할수 있다.

pvm_mcast가 성과적으로 완료하면 info > 0이며 오유가 발생하면 info < 0으로 된다. 수신프로쎄스는 pvm_recv나 pvm_nrecv를 호출하여 다중통신에서 자기 부분만을 수신할수 있다. 통보가 수신프로쎄스에 안전하게 도착하자마자 송신프로쎄스는 자기처리를 계속한다.

이것은 수신프로쎄스가 통보를 정확히 접수하였다는것을 확인할때까지 송신처리기에서의 계산이 중지하는 동기식통신과는 대조적이다.

pvm_mcast는 지적한 과제들이 실행하는 pvmd들을 결정한다. 그 다음 통보들을 이 pvmd들에 넘긴다. 다음 pvmd들은 이 통보들을 국부과제들에 분배한다. 대다수의 다중처리기제작자들은 다중통신을 지원하지 않는다.

일반적으로는 다중처리기에 있는 모든 리용자프로쎄스들에 방송통신을 제공한다. 이러한 제한성으로 하여 pvm_mcast는 모든 병렬가상기계프로쎄스들에 방송하는 특수한 경우를 제외하고는 다중처리기들에서 효과적인 통신수법으로 되지 못한다.

실례프로그람

C:

```
info = pvm_initsend( PvmDataRaw );
info = pvm_pkint( array, 10, 1 );
 msgtag = 5 ;
 info = pvm_mcast( tids, ntask, msgtag );
```

포트란:

```
CALL PVMFINITSEND(PVMDEFAULT)
CALL PVMFPACK( REAL4, DATA, 100, 1,
                    INFO )
CALL PVMFMCAST( NPROC, TIDS, 5, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	msgtag < 0 이다.
PvmSysErr	pvmd가 응답하지 않는다.
PvmNoBuf	송신완충기가 없다.

pvmfmkbuf()　　　　　　　　pvm_mkbuf()

새로운 송신완충기를 창조한다.

문법

C:

```
int bufid = pvm_mkbuf( int encoding )
```

포트란:

```
call pvmfmkbuf( encoding, bufid )
```

변수

encoding – 완충기의 부호화수법을 지적한다.

C에는 다음의 추가선택들이 있다.

부호	값	의미
PvmDataDefault	0	XDR
PvmDataRaw	1	부호화를 하지 않는다.
PvmDataInPlace	2	왼쪽자료배치

bufid – 귀환된 통보완충기식별자, 오유가 생기면 0보다 작은값이 설정된다.

설명

pvm_mkbuf는 새로운 통보완충기를 창조하며 그의 부호화상태를 encoding으로 설정한다. pvm_mkbuf가 성과적으로 완료하면 bufid는 새로운 완충기에 대한 식별자로 되며 오유가 생기면 bufid < 0로 된다. 기정으로는 XDR부호화가 리용된다. 비록 체계가 이종이라고 해도 리용자가 가상기계에 대한 지식을 얻을수 있도록 여러가지 부호화 추가선택들이 리용된다.

실례로 다음에 통보를 보낼 목적처리기에서 리용하는 자료변환형식이 원천처리기의 자료변환형식과 같다는것을 리용자가 사전에 알고있다면 PvmDataRaw부호화를 리용하여 부호화로 인한 비용을 줄일수 있다.

PvmDataInPlace부호화방법을 리용하면 압축시에 자료를 왼쪽에서부터 배치할수 있다. 통보완충기는 다만 보내야 할 항목들의 크기와 지적자만을 보관하고있다.

pvm_send를 호출할 때 항목들은 리용자의 기억기밖으로 직접 복사된다. 이 추가선택은 통보를 복사하는 회수를 줄인다. PvmDataIn Place 는 PVM 3.2에서는 실현할수 없다.

리용자가 다중통보완충기들을 리용하려고 한다면 pvm_mkbuf를 pvm_freebuf와 함께 리용할수 있다. 통보가 전송되고 통보완충기가 더이상 필요없으면 pvm_freebuf를 써서 송신완충기를 해방한다.

수신완충기들은 pvm_recv와 pvm_nrecv를 리용하면 자동적으로 창조된다. 리용자들은 완충기를 해방하기전에 pvm_setrbuf를 리용하여 확보하여야 한다.

일반적으로 다중송수신완충기들은 필요없다. 리용자들은 pvm_initsend를 리용하여 기정의 송신완충기들을 재설정한다.

다중완충기 들이 쓸모있는 경우도 있다.

실례로 병렬가상기계를 리용하는 서고 혹은 시각대면부들에서와 실행하는 병렬가상기계응용프로그람들과 호상작용하지만 응용프로그람 자체의 통신과는 간섭하려고 하지 않는 경우에 다중완충기가 효과적이다.

다중완충기들을 리용할 때 일반적으로 압축하는 매개 통보들에 대하여 완충기들이 설정된다.

실례프로그람

C:

```
bufid = pvm_mkbuf( PvmDataRaw );
/* send the message */
info = pvm_freebuf( bufid );
```

포트란:

```
CALL PVMFMKBUF(PVMDEFAULT, MBUF)
* SEND MESSAGE HERE
CALL PVMFFREEBUF( MBUF, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	무효한 부호화 값
PvmNoMem	malloc가 실패하였다.

pvmfmstat() pvm_mstat()

가상기계에 있는 처리기의 상태를 돌려준다.

문법

C:

```
int mstat = pvm_mstat( char *host )
```

포트란:

```
call pvmfmstat( host, mstat )
```

변수

host – 처리기이름을 가지고있는 문자렬

mstat – 가상기계의 상태:

값	의미
PvmOk	성공
PvmNoHost	가상기계에 그 처리기가 없다.
PvmHostFail	처리기에 접근 불가능

설명

pvm_mstat는 실행하고있는 병렬가상기계프로쎄스와 관련한 처리기 host의 상태 mstat를 돌려준다. 이 함수를 리용하여 어느 처리기가 실패하였는가, 가상기계를 재구성할 필요는 없는가를 검사할수 있다.

실례프로그람

C:

```
mstat = pvm_mstat( "msr.ornl.gov" );
```

포트란:

```
CALL PVMFMSTAT( 'msr.ornl.gov', MSTAT )
```

오유:

이름	가능한 경우
PvmSysErr	pvmd가 응답하지 않는다.
PvmNoHost	가상기계에 이 처리기가 없다.
PvmHostFail	처리기접근불가능

프로세스의 tid를 돌려준다.

문법

C:

```
int tid = pvm_mytid( void )
```

포트란:

```
call pvmfmytid( tid )
```

변수

tid – 호출한 병렬가상기계프로세스의 옹근수식별자를 돌려준다. 오유가 발생하면 0보다 작은 값이 설정된다.

설명

이 함수를 처음 호출하면 먼저 이 프로세스를 병렬가상기계에 등록하며 이 프로세스가 pvm_spawn에 의해 창조된것이 아니라면 유일한 tid를 생성한다.

pvm_mytid는 호출한 프로세스의 tid를 돌리며 응용프로그람내에서 여러번 호출할수 있다.

만일 이 함수를 호출하기전에 과제가 체계에 등록되여있지 않다면 임의의 병렬가상기계체계호출로 과제를 등록할수 있다.

tid는 국부 pvmd가 창조하는 32bit정의옹근수이다. 32bit구역은 가상기계에서의 그의 위치(pvmd의 주소), 프로쎄스가 다중처리기에서 실행하는 경우 CPU번호, 프로쎄스id마당과 같은 여러가지 정보들을 가지는 마당으로 갈라진다. 이 정보는 병렬가상기계에서만 리용하며 응용프로그람에서는 리용할수 없다. 만일 응용프로그람이 pvm_mytid를 호출하기전에 병렬가상기계가 시동하지 않았으면 tid<0으로 된다.

실례프로그람

C:

```
tid = pvm_mytid( );
```

포트란:

```
CALL PVMFMYTID( TID )
```

오유:

이름	가능한 경우
PvmSysErr	pvmd가 응답하지 않는다.

pvmfnotify() pvm_notify()

처리기의 실패와 같은 병렬가상기계사건들을 통지할것을 요구한다.

문법

C:

```
int info = pvm_notify( int what, int msgtag,
                       int cnt, int *tids )
```

포트란:

```
call pvmfnotify( what, msgtag, cnt, tids, info )
```

변수

what – 통지할 사건을 지적하는 옹근수
다음의 추가선택들이 있다.

값	의미
PvmTaskExit	통지가 없다.
PvmHostDelete	처리기를삭제하면 통지한다.
PvmHostAdd	처리기를 추가하면 통지한다.

msgtag – 통지에 리용되는 통보표적

cnt – PvmTaskExit와 PvmHostDelete를 위한 tids배렬의 길이를 지정하는 옹근수, PvmHostAdd에 대해서는 통지회수를 지적한다.

tids – 통지할 과제목록 혹은 pvmd의 tid들을 가지고있는 길이가 ntasks인 옹근수배렬, PvmHostAdd추가선택인 경우에는 이것이 빈 배렬로 된다.

info – 귀환 상태값, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_notify는 병렬가상기계가 어떤 사건을 검출한 경우에 이 함수를 호출한 프로쎄스에 통지할것을 요구한다.

통지요구에 응답하여 병렬가상기계는 몇개의 통보들을 호출한 과제에 보낸다. 통보들에는 이 통보들이 통지문이라는것을 표시하는 표적을 붙인다. tids배렬은 TaskExit나 HostDelete를 리용할 때 누가 이것을 조종하는가를 지적한다.

HostAdd를 리용할 때는 아무것도 요구하지 않는다. 만일 요구하면 pvm_config와 pvm_tasks를 리용하여 과제와 pvmd의 id를 얻을수 있다. 통지문 통보들은 다음의 형식을 가지고있다.

PvmTaskExit – 매 tid에 대하여 한개의 통지문 통보가 요구된다. 통보본체는 완료된 과제의 단일한 tid를 가지고있다.

PvmHostDelete - 매 tid에 대하여 한개의 통지문 통보가 요구된다. 통보체는 완료된 pvmd의 단일한 tid를 가지고있다.

PvmHostAdd – cnt개수까지의 통보들을 보낸다. 통보는 옹근수 count를 포함하며 다음에 새로운 pvmd들에 대한 tids목록을 가지고있다.
나머지 PvmHostAdd통보들의 계수기는 다음 pvm_notify를 호출하여 갱신한다.

통지문 통보들에 있는 tids들은 옹근수로 압축한다. 호출한 과제들은 지적한 msgtag를 가진 통보들을 수신하고 그에 따르는 처리를 진행할수 있다.

실례프로그람

C:

```
   info = pvm_notify( PvmHostAdd, 9999, 1, dummy );
```

포트란:

```
      CALL PVMFNOTIFY( PVMHOSTDELETE, 1111,
                       NPROC, TIDS, INFO )
```

오유:

이름	가능한 경우
PvmSysErr	pvmd가 응답지 않는다.
PvmBadParam	무효한 변수값

pvmfnrecv()　　　　　　　　pvm_nrecv()

msgtag라는 표식을 가진 비봉쇄통보가 있는가를 검사한다.

문법

C:

```
int bufid = pvm_nrecv( int tid, int msgtag )
```

포트란:

```
call pvmfnrecv( tid, msgtag, bufid )
```

변수

tid – 리용자가 준 송신프로쎄스의 과제식별자(-1은 임의의 과제임을 표시한다.)

msgtag – 리용자가 준 통보표적, msgtag >= 0이여야 한다.

bufid – 새로운 능동수신완충기의 식별자를 돌리는 옹근수, 오유가 발생하면 0보다 작은 값이 설정된다.

설명

pvm_nrecv는 msgtag라는 표식을 가진 통보가 tid로부터 도착했는가를 검사한다. 일치하는 통보가 도착하면 pvm_nrecv는 통보를 직접 능동인 수신완충기에 배치하며 현재의 수신완충기를 지운다. 그리고 bufid에 통보완충기를 돌린다.

만일 요구하는 통보가 도착하지 않으면 pvm_nrecv는 bufid에 0을 돌린다. 오유가 생기면 bufid < 0로 된다.

msgtag나 tid에 –1을 설정하면 임의의것을 준다. 이것은 리용자가 다음의 추가선택을 리용하게 한다.

만일 tid = -1이고 msgtag 를 리용자가 정의하였다면 pvm_nrecv는 msgtag와 일치하는 임의의 프로쎄스로부터 통보를 받는다.

만일 msgtag = -1이고 tid를 리용자가 정의하였다면 pvm_nrecv는 프로쎄스 tid가 보내는 임의의 통보를 접수한다.

만일 tid = -1이고 msgtag = -1이면 pvm_nrecv는 임의의 프로쎄스로부터 오는 임의의 통보를 접수한다.

pvm_nrecv는 통보가 국부 pvmd에 아직 도착하지 않았다는 통보 혹은 정보를 가지고 즉시 귀환된다는 의미에서 비봉쇄적이다. pvm_nrecv는 여러번 호출하여 지적한 통보가 도착했는가를 검사할수 있다.

추가적으로 pvm_recv는 만일 응용프로그람이 자료를 접수하기전에 자기작업을 끝낸다면 같은 통보에 대하여 또 호출할수 있다. pvm_nrecv로 통보를 접수하면 통보에 있는 자료는 풀기함수를 리용하여 리용자의 기억구역에 풀린다.

병렬가상기계모델은 통보의 순서를 담보한다. 만일 과제 1이 통보 A를 과제 2에 보내고 그 다음 과제 1이 통보 B를 과제 2에 보내면 통보 A는 통보 B보다 과제 2에 먼저 도착한다. 더우기 두 통보들이 과제 2가 접수하기전에 도착해도 항상 통보 A를 먼저 접수한다.

실례프로그람

C:

```
        tid = pvm_parent();
        msgtag = 4 ;
        arrived = pvm_nrecv( tid, msgtag );
        if ( arrived > 0 )
            info = pvm_upkint( tid_array, 10, 1 );
        else
           /* go do other computing */
```

포트란:

```
       CALL PVMFNRECV( -1, 4, ARRIVED )
       IF ( ARRIVED .GT. 0 ) THEN
      CALL PVMFUNPACK( INTEGER4, TIDS, 25, 1,
                              INFO )
      CALL PVMFUNPACK( REAL8, MATRIX, 100,
                               100, INFO )
```

```
ELSE
   * GO DO USEFUL WORK
   ENDIF
```

오유:

이름	가능한 경우
PvmBadParam	무효한 tid 값 혹은 msgtag
PvmSysErr	pvmd가 응답하지 않는다.

pvmfpack() pvm_pk*()

지정된 자료형배렬로 능동통보완충기를 압축한다.

문법

C:

```
int info = pvm_packf( const char *fmt, ... )
int info = pvm_pkbyte( char *xp, int nitem, int stride )
int info = pvm_pkcplx( float *cp, int nitem, int stride )
int info = pvm_pkdcplx( double *zp, int nitem,
                        int stride)
int info = pvm_pkdouble(double *dp, int nitem,
                        int stride )
int info = pvm_pkfloat( float *fp, int nitem, int stride )
int info = pvm_pkint( int *ip, int nitem, int stride )
int info = pvm_pkuint( unsigned int *ip, int nitem,
                       int stride )
int info = pvm_pkushort( unsigned short *ip,
                         int nitem, int stride )
int info = pvm_pkulong( unsigned long *ip, int nitem,
                        int stride )
int info = pvm_pklong( long *ip, int nitem, int stride )
int info = pvm_pkshort( short *jp, int nitem, int stride )
int info = pvm_pkstr( char *sp )
```

포트란:

```
call pvmfpack( what, xp, nitem, stride, info )
```

변수

fmt – 압축방법을 지적하는 인쇄가능한 형식

nitem – 압축할 항목들의 전체 개수

stride – 항목들을 압축할 때 리용하는 걸음, 실례로 pvm_pkcplx에서 stride=2이면 모든 복소수들이 압축된다.

xp – 바이트블로크들의 시작에로의 지적자, 임의의 자료형이 될수 있지만 대응하는 풀기자료형과 일치해야 한다.

cp – 적어도 nitem*stride의 길이를 가지는 배렬

zp – 적어도 nitem*stride의 길이를 가지는 배정확도배렬

dp – 적어도 nitem*stride의 길이를 가지는 배정확도실수배렬

fp – 적어도 nitem*stride의 길이를 가지는 실수배렬

ip – 적어도 nitem*stride의 길이를 가지는 옹근수배렬

jp – 적어도 nitem*stride의 길이를 가지는 integer*2배렬

sp – null로 끝나는 문자렬에로의 지적자

what – 압축할 자료의 형을 지적하는 옹근수

what		추가선택	
STRING	0	REAL4	4
BYTE1	1	COMPLEX8	5
INTEGER2	2	REAL8	6
INTEGER4	3	COMPLEX16	7

info – 귀환 코드, 오유가 생기면 0보다 작은 값이 설정된다.

설명

매개 pvm_pk*함수들은 주어진 자료형의 배렬들을 능동송신완충기에 압축한다. 매 함수들에서 변수는 압축할 첫 항목들에로의 지적자이며

nitem은 이 배렬에서 압축할 항목들의 전체 개수, stride는 압축할 때 리용되는 걸음이다.

한가지 례외는 pvm_pkstr()이며 여기서는 정의에 의하여 null로 끝나는 문자렬을 압축하므로 nitem 이나 stride같은 변수가 필요없다.

포트란함수 pvmfpack(STRING, ...)에서는 nitem이 문자렬에 있는 문자들의 개수라고 가정하며 stride=1로 가정한다.

압축이 성공하면 info = 0, 오유가 발생하면 info < 0로 된다.

단일변수인 경우에는 nitem=1, stride=1로 설정하여 압축할수 있다. 구조체들은 한번에 한개의 자료형을 압축해야 한다.

pvm_packf()는 인쇄가능한 형식의 표기를 리용하여 자료를 무엇으로 또 어떻게 압축하여 송신완충기에 보관하는가를 지적한다.

모든 변수는 stride와 count 가 지적되면 주소로서 넘어가며 그외 다른 경우에 변수는 값으로 넘어간다.

BNF와 류사한 형식적서술법이 리용된다.

format : null | init | format fmt

init : null | '%' '+'

fmt : '%' count stride modifiers fchar

fchar : 'c' | 'd' | 'f' | 'x' | 's'

count : null | [0-9]+ | '*'

stride : null | '.' ([0-9]+ | '*')

modifiers : null | modifiers mchar

mchar : 'h' | 'l' | 'u'

형식:

+ 는 initsend를 의미하며 param목록에 있는 int와 일치해야 한다.

c 압축/풀기할 바이트를 지적한다.

d 옹근수

f 류점수

x 배정확도류점수

s 문자렬

략자:

h short (int)

l long (int, float, complex float)
u unsigned (int)
'*' count 나 stride는 param목록에 있는 int와 일치해야 한다.

실례프로그람
C:
```
        info = pvm_initsend( PvmDataDefault );
        info = pvm_pkstr( "initial data" );
        info = pvm_pkint( &size, 1, 1 );
        info = pvm_pkint( array, size, 1 );
        info = pvm_pkdouble( matrix, size*size, 1 );
        msgtag = 3 ;
        info = pvm_send( tid, msgtag );
```
포트란:
```
       CALL PVMFINITSEND(PVMRAW, INFO)
       CALL PVMFPACK( INTEGER4, NSIZE, 1, 1,
                      INFO )
       CALL PVMFPACK( STRING, 'row 5 of NXN
                      matrix', 19, 1, INFO )
       CALL PVMFPACK( REAL8, A(5,1), NSIZE,
                      NSIZE , INFO )
       CALL PVMFSEND( TID, MSGTAG, INFO )
```
오유:

이름	가능한 경우
PvmNoMem	malloc가 실패, 통보완충기크기가 이 처리기에서 리용할수 있는 기억기를 초과한다.
PvmNoBuf	압축에 필요한 능동인 송신완충기가 없다.

pvmfparent()   pvm_parent()

호출한 프로세스의 어미프로세스에 대한 tid를 돌려준다.

문법

C:

```
int tid = pvm_parent( void )
```

포트란:

```
call pvmfparent( tid )
```

변수

tid – 어미프로쎄스의 과제식별자, 만일 어미과제가 없으면 tid = PvmNoParent

설명

pvm_parent는 호출한 프로쎄스를 생성한 프로쎄스의 tid를 돌려준다. 만일 어미과제가 없으면 tid = PvmNoParent이다.

실례프로그람

C:

```
tid = pvm_parent();
```

포트란:

```
CALL PVMFPARENT( TID )
```

오유:

이름	가능한 경우
PvmNoParent	호출한 프로쎄스는 pvm_spawn으로 생성된 프로쎄스가 아니다.

pvmfperror() pvm_perror()

마지막으로 호출한 병렬가상기계호출의 오유상태를 돌린다.

문법

C:

```
int info = pvm_perror( char *msg )
```

포트란:

```
call pvmfperror( msg, info )
```

변수

msg – 마지막 병렬가상기계호출의 오유통보에 첨부되는 리용자가 제공하는 문자렬

info – 귀환코드, 오유가 생긴 경우 0보다 작은 값이 설정된다.

설명

pvm_perror는 마지막 병렬가상기계호출에 대한 오유통보를 돌려준다. 리용자는 msg를 리용하여 오유통보에 보충적인 정보들, 실례로 그의 위치 등을 첨부할수 있다.

모든 표준출구 및 표준오유통보들은 주 pvmd가 실행하는 처리기의 /tmp/pvml.<uid>파일에 보관된다.

실례프로그람

C:

```
if ( pvm_send( tid, msgtag ))
pvm_perror();
```

포트란:

```
CALL PVMFSEND( TID, MSGTAG )
IF( INFO .LT. 0 )
CALL PVMFPERROR( 'Step 6', INFO )
```

오유:

오유없음

pvmfprecv() pvm_precv()

완충기안에로 통보를 직접 수신한다.

문법

C:

```
int info = pvm_psend( int tid, int msgtag,
                      char *buf, int len, int datatype, int
                      atid, int atag, int alen )
```

포트란:

```
call pvmfpsend( tid, msgtag, buf, len, datatype,
                atid, atag, alen, info )
```

변수
tid – 송신프로쎄스의 과제식별자
msgtag – 통보표적, msgtag >= 0이여야 한다.
buf – 통보를 수신할 완충기에로의 지적자
len – 완충기의 길이
datatype – buf가 지적하는 자료의 형
atid – 송신자의 실제 TID
atag – 실제적인 통보표적
atid – 통보의 실제길이
info – 상태코드, 오유가 생기면 0보다 작은 값이 설정된다.

설명
pvm_recv는 msgtag라는 표식을 가진 통보가 tid로부터 도착할 때까지 프로쎄스를 봉쇄한다. pvm_precv는 다음 준비된 통보완충기 buf에 통보를 넣는다.

pvm_precv는 pvm_psend, pvm_send, pvm_mcast, 혹은 pvm_bcast가 보낸 통보를 접수할수 있다. msgtag = -1이든가 tid = -1 이면 임의의 통보를 접수한다.

이때 리용자는 아래의 추가선택들중 임의의 추가선택을 리용할수 있다.

tid = -1이고 msgtag를 리용자가 정의하였다면 pvm_recv는 임의의 프로쎄스로부터 msgtag를 가진 통보를 접수할수 있다.

msgtag = -1이고 tid 를 리용자가 정의하였다면 pvm_recv는 프로쎄스 tid가 보내는 임의의 통보를 접수한다.

tid = -1이고 msgtag = -1이면 pvm_recv는 임의의 프로쎄스로부터 임의의 통보를 접수한다.

C에서 datatype변수는 전송하는 자료의 형에 따라 다음의 변수중 하나로 된다.

datatype	자료형
PVM_STR	string
PVM_BYTE	byte
PVM_SHORT	short

PVM_INT	int
PVM_FLOAT	real
PVM_CPLX	complex
PVM_DOUBLE	double
PVM_DCPLX	double complex
PVM_LONG	long integer
PVM_USHORT	unsigned short int
PVM_UINT	unsigned int
PVM_ULONG	unsigned long int

포트란에서는 풀기에서와 같은 자료형들이 리용된다. 병렬가상기계 모형은 통보의 순서를 담보한다. 만일 과제 1이 통보 A를 과제 2에 보내고 다음 과제 1이 통보 B를 과제 2 에 보내면 통보 A는 통보 B보다 과제 2에 먼저 도착한다.

더우기 두 통보들이 과제 2가 접수하기전에 도착해도 항상 통보 A를 먼저 접수한다. pvm_recv는 봉쇄된다. 이것은 리용자가 지적한 tid와 msgtag를 가진 통보가 국부 pvmd에 도착할 때까지 이 함수가 대기하고있는다는것을 의미한다. 만일 통보가 이미 도착하였다면 pvm_recv는 통보와 함께 즉시 귀환한다. pvm_precv는 현재 수신통보완충기의 상태에 영향을 주지 않는다.

실례프로그람

C:

```
info = pvm_precv( tid, msgtag, array, cnt,
                  PVM_FLOAT, &src, &atag, &acnt );
```

포트란:

```
CALL PVMFPRECV( -1, 4, BUF, CNT, REAL4,
                SRC, ATAG, ACNT, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	무효한 tid 혹은 msgtag
PvmSysErr	pvmd가 응답하지 않는다.

통보가 도착했는가를 검사한다.

문법

C:

```
int bufid = pvm_probe( int tid, int msgtag )
```

포트란:

```
call pvmfprobe( tid, msgtag, bufid )
```

변수

tid – 리용자가 주는 송신프로쎄스의 프로쎄스 id

msgtag – 리용자가 주는 통보의 표적, msgtag >= 0 이여야 한다.

bufid – 새로운 능동수신완충기식별자의 값, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_probe는 msgtag라는 표식이 붙은 통보가 tid로부터 도착했는가를 검사한다. 일치하는 통보가 도착하면 pvm_probe는 bufid에 완충기식별자를 넘겨준다. 이 bufid는 pvm_bufinfo호출에서 리용하여 그의 원천지와 길이가 같은 통보에 대한 정보를 결정하는데 리용된다.

만일 요구하는 통보가 도착하지 않으면 pvm_probe는 bufid=0을 돌린다. 오유가 발생하면 bufid < 0으로 된다. msgtag = -1이든가 tid = -1 이면 임의의 통보를 접수한다.

이때 리용자는 아래의 추가선택들중 임의의 추가선택을 리용할수 있다. tid = -1 이고 msgtag를 리용자가 정의하였다면 pvm_probe는 임의의 프로쎄스로부터 msgtag를 가진 통보를 접수할수 있다. msgtag = -1이고 tid 를 리용자가 정의하였다면 pvm_probe는 프로쎄스 tid가 보내는 임의의 통보를 접수한다.

tid = -1이고 msgtag = -1이면 pvm_probe는 임의의 프로쎄스로부터 임의의 통보를 접수한다. 리용자는 pvm_probe는 여러번 호출하여 주어진 통보가 도착하였는가를 검사할수 있다.

통보가 도착하면 풀기함수를 리용하여 통보를 리용자의 기억구역에 풀기전에 pvm_recv를 먼저 호출하여야 한다.

실례프로그람

C:

```
tid = pvm_parent();
msgtag = 4 ;
arrived = pvm_probe( tid, msgtag );
if ( arrived > 0 )
info = pvm_bufinfo( arrived, &len, &tag, &tid );
else
/* 다른 계산을 진행 */
```

포트란:

```
CALL PVMFPROBE( -1, 4, ARRIVED )
IF ( ARRIVED .GT. 0 ) THEN
CALL PVMFBUFINFO( ARRIVED, LEN, TAG,
                  TID, INFO )
ELSE
* GO DO USEFUL WORK
ENDIF
```

오유:

이름	가능한 경우
PvmBadParam	무효한 tid 혹은 msgtag
PvmSysErr	pvmd가 응답하지 않는다.

pvmfpsend() pvm_psend()

한번의 호출로 자료를 압축하고 전송한다.

문법

C:

```
    int info = pvm_psend( int tid, int msgtag, char
```

*buf, int len, int datatype)

포트란:

call pvmfpsend(tid, msgtag, buf, len, datatype, info)

변수

tid – 목적지 프로세스의 과제식별자

msgtag – 리용자가 지적한 통보표적, msgtag >= 0 이다.

buf – 송신할 완충기에로의 지적자

len – 완충기의 길이

datatype – buf가 지적하는 자료의 형

info – 상태코드, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_psend는 길이가 len이고 그의 자료형이 datatype인 완충기 buf에로의 지적자를 주며 이 자료를 지적된 병렬가상기계과제에 직접 전송한다. pvm_psend자료는 pvm_precv, pvm_recv, pvm_trecv, 혹은 pvm_nrecv로 접수할수 있다.

msgtag는 통보의 내용을 표시하는데 리용할수 있다. pvm_send는 성공하면 info = 0이며 오유가 생기면 info < 0으로 된다. pvm_psend는 비동기적이다.

송신프로쎄스에서 연산은 통보가 수신프로쎄스에 안전하게 도착하자마자 계속하여 진행된다. 이 방식은 동기통신과는 대조적이다. C에서 datatype변수는 다음의 값들중의 하나일수 있다.

datatype	자료형
PVM_STR	string
PVM_BYTE	byte
PVM_SHORT	short
PVM_INT	int
PVM_FLOAT	real
PVM_CPLX	complex
PVM_DOUBLE	double
PVM_DCPLX	double complex

PVM_LONG	long integer
PVM_USHORT	unsigned short int
PVM_UINT	unsigned int
PVM_ULONG	unsigned long int

포트란에서는 압축을 할 때 리용한 자료형과 같은 형의 자료형들이 리용된다. 병렬가상기계모형은 통보의 순서를 보존한다.

만일 과제 1이 통보 A를 과제 2에 보내고 그 다음 과제 1이 통보 B를 과제 2 에 보내면 통보 A는 통보 B보다 과제 2에 먼저 도착한다. 더우기 두 통보들이 과제 2가 접수하기전에 도착해도 항상 통보 A를 먼저 접수한다.

pvm_psend는 현재 송신을 위해 준비된 다른 통보완충기의 상태에는 영향을 주지 않는다.

실례프로그람

C:

```
        info = pvm_psend( tid, msgtag, array, 1000,
                          PVM_FLOAT );
```

포트란:

```
        CALL PVMFPSEND( TID, MSGTAG,
                  BUF, CNT, REAL4, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	무효한 tid 혹은 msgtag
PvmSysErr	pvmd가 응답하지 않는다.

pvmfpstat() pvm_pstat()

지적된 프로세스의 상태를 돌려준다.

문법

C:

int status = pvm_pstat(tid)

포트란:

call pvmfpstat(tid, status)

변수

tid – PVM프로쎄스의 과제식별자

status – tid로 지적된 병렬가상기계프로쎄스의 상태

과제가 실행하고있으면 상태는 PvmOk이며 그러한 과제가 없으면 PvmNoTask, tid가 틀리면 PvmBadParam로 된다.

설명

pvm_pstat는 tid로 지적된 프로쎄스의 상태를 돌려준다.

실례프로그람

C:

```
tid = pvm_parent();
status = pvm_pstat( tid );
```

포트란:

```
CALL PVMFPARENT( TID )
CALL PVMFPSTAT( TID, STATUS )
```

오유:

이름	가능한 경우
PvmBadParam	변수가 틀렸다.
PvmSysErr	pvmd가 응답하지 않는다.
PvmNoTask	과제가 실행하지 않는다.

pvmfrecv() pvm_recv()

통보를 수신한다.

문법

C:

```
int bufid = pvm_recv( int tid, int msgtag )
```

포트란:

call pvmfrecv(tid, msgtag, bufid)

변수

tid – 리용자가 제공한 송신측 프로쎄스의 과제식별자

msgtag – 리용자가 제공하는 통보의 표적, msgtag >= 0이다. 리용자는 이 변수의 내용을 보고 여러가지 통보들을 구별한다.

bufid – 새로운 능동수신완충기의 식별자를 돌린다. 오유가 생기면 0보다 작은 값을 돌린다.

설명

pvm_recv는 msgtag라는 표식을 가진 통보가 tid로부터 도착할 때까지 봉쇄한다. pvm_recv는 다음 통보를 새로운 능동수신완충기에 배치하는데 이때 현재의 수신완충기를 지운다. msgtag = -1 이든가 tid = -1이면 임의의 통보를 접수한다. 이때 리용자는 아래의 추가선택들중 임의의 추가선택을 리용할수 있다.

tid = -1 이고 msgtag를 리용자가 정의하였다면 pvm_recv는 임의의 프로쎄스로부터 msgtag를 가진 통보를 접수할수 있다.

msgtag = -1이고 tid 를 리용자가 정의하였다면 pvm_recv는 프로쎄스 tid가 보내는 임의의 통보를 접수한다.

tid = -1이고 msgtag = -1이면 pvm_recv는 임의의 프로쎄스로부터 임의의 통보를 접수한다. 병렬가상기계모델은 통보의 순서를 보존한다.

만일 과제 1이 통보 A를 과제 2에 보내고 그 다음 과제 1이 통보 B를 과제 2 에 보내면 통보 A는 통보 B보다 과제 2에 먼저 도착한다. 더우기 두 통보들이 과제 2가 접수하기전에 도착해도 항상 통보 A를 먼저 접수한다.

pvm_recv가 성공하면 bufid에는 새로운 능동수신완충기의 값이 보관되며 실패하면 bufid < 0으로 된다.

pvm_recv는 봉쇄한다. 이것은 리용자가 지적한 tid와 msgtag와 일치하는 통보가 pvmd로부터 도착할 때까지 이 함수가 기다린다는것을 의미한다.

통보가 이미 도착하였으면 pvm_recv는 통보와 함께 즉시 귀환한다. 일단 pvm_recv가 돌아오면 통보에 있는 자료는 풀기함수를 리용하여 리용자의 기억기에 풀어진다.

실례프로그람
C:
```
tid = pvm_parent();
msgtag = 4 ;
bufid = pvm_recv( tid, msgtag );
info = pvm_upkint( tid_array, 10, 1 );
info = pvm_upkint( problem_size, 1, 1 );
info = pvm_upkfloat( input_array, 100, 1 );
```

포트란:
```
    CALL PVMFRECV( -1, 4, BUFID )
    CALL PVMFUNPACK( INTEGER4, TIDS, 25, 1,
                         INFO )
    CALL PVMFUNPACK( REAL8, MATRIX, 100,
                         100, INFO )
```
오유:

이름	가능한 경우
PvmBadParam	무효한 tid 혹은 msgtag <-1
PvmSysErr	pvmd가 응답하지 않는다.

pvm_recvf()

통보를 접수하기 위하여 비교함수들을 다시 정의한다.

문법
C:
```
int (*old)() = pvm_recvf( int (*new)( int bufid, int tid, int tag ))
```
포트란:
　　없음
변수
tid – 리용자가 제공한 송신프로세스의 과제식별자
tag – 리용자가 제공한 통보표적
bufid – 통보완충기식별자

설명

pvm_recvf는 pvm_recv와 pvm_nrecv함수들에서 리용하는 비교함수들을 정의한다.

pvm_recvf는 리용자가 제공한 비교함수를 설정하여 수신하는 통보들을 평가한다. 기정의 비교함수는 모든 오는 통보들과 관련한 원천 및 통보표적들을 평가한다.

pvm_recvf는 그러한 함수들의 기능을 파악하고있고 기정의것보다 더 복잡한 통보문맥들을 통신하려고 하는 C프로그람작성자들을 위하여 나왔다.

pvm_recvf는 기정의 일치함수라면 0 을 돌려주며 다른 경우 일치함수를 돌려준다. 일치함수는 다음의 값을 돌린다.

값	취하는 동작
< 0	오유코드를 가지고 즉시 귀환한다.
0	이 통보를 받지 않는다.
1	이 통보를 받으며 나머지는 주사하지 않는다.
> 1	주사후에 가장 높은 급의 통보를 받는다.

실례프로그람

```
#include "pvm.h"
static int foundit = 0;

static int
foo_match(mid, tid, code)
        int mid;
        int tid;
        int code;
{
int t, c, cc;
if ((cc = pvm_bufinfo(mid, (int *)0, &c, &t)) < 0)
return cc;
if ((tid == -1 || tid == t)
```

```
&& (code == -1 || code == c))
foundit = 1;
return 0;
}

int probe(src, code)
   {
int (*omatch)();
int cc;
omatch = pvm_recvf(foo_match);
foundit = 0;
if ((cc = pvm_nrecv(src, code)) < 0)
    return cc;
pvm_recvf(omatch);
return foundit;
}
```

오유:

오유없음

pvmfreduce() pvm_reduce()

지적한 그룹의 성원들에서 축소연산을 수행한다.

문법

C:

```
     int info = pvm_reduce( void (*func)(),
               void *data, int count, int datatype,
               int msgtag, char *group, int root)
```

포트란:

```
     call pvmfreduce( func, data, count, datatype,
                          msgtag, group, root, info )
```

변수

func – 대역자료우에서 진행하는 연산들을 정의하는 함수
미리 정의된 함수들은 PvmMax, PvmMin, PvmSum,
그리고 PvmProduct이다. 리용자들은 자기자체의
함수들을 정의할수도 있다.

data – 국부값 배렬의 시작주소에로의 지적자, 귀환할 때 뿌리에 있는 자료배렬은 그룹에서의 축소연산결과와 함께 중복하여 쓴다.

count – 자료배렬에 있는 요소들의 개수를 지적하는 옹근수

datatype – 자료배렬에 있는 마디들의 형을 지적하는 옹근수

msgtag – 리용자가 제공하는 통보표적, msgtag >= 0이여야 한다.

group – 현존 그룹의 그룹이름

root – 결과를 얻는 그룹성원의 실체개수

info – 상태코드, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_reduce() 는 그룹에 있는 모든 과제들에서 최대값, 최소값얻기, 더하기와 같은 연산을 수행한다.

모든 그룹성원들은 자기의 국부자료와 함께 pvm_reduce()를 호출하며 축소연산의 결과는 리용자가 지적한 뿌리과제 root에 나타난다. 뿌리과제는 그룹에 있는 그의 실체번호로 나타난다.

병렬가상기계는 다음과 같은 미리 정의한 대역함수들을 func에 지적할수 있다.

PvmMin
PvmMax
PvmSum
PvmProduct

PvmMax와 PvmMin는 자료형이 byte, short, integer, long, float, double, complex, double complex인 자료들에 대하여 수행한다.

PvmSum과 PvmProduct는 자료형이 short, integer, long, float, double, complex, 그리고 double complex인 자료들에 대하여 수행한다. C와 포트란에는 다음의 자료형들이 미리 정의되여있다.

C 자료형	포트란 자료형
PVM_BYTE	BYTE1
PVM_SHORT	INT2
PVM_INT	INT4
PVM_FLOAT	REAL4
PVM_CPLX	COMPLEX8
PVM_DOUBLE	REAL8
PVM_DCPLX	COMPLEX16
PVM_LONG	

리용자가 정의한 함수도 func에서 리용할수 있는데 그 형식은 아래와 같다.

C:

```
void func(int *datatype, void *x, void *y,int *num, int *info)
```

포트란:

```
call func(datatype, x, y, num, info)
```

func는 축소연산에서 리용하는 기본기능이다. x 와 y 는 num개의 입구점들을 가지고있는 datatype형의 자료배렬이다. 변수 datatype와 info는 우에서 지적한것과 같다. 변수 x와 num은 우의 data와 count값과 대응된다. 변수 y는 수신한 값들을 가지고있다.

주의:

pvm_reduce()는 봉쇄하지 않는다. 만일 과제가 pvm_reduce를 호출하고 뿌리과제가 pvm_reduce를 호출하기전에 그룹을 리탈한다면 유가 생긴다. 현재 알고리듬은 아주 단순하면서도 믿음직하다.

실례프로그람

C:

```
info = pvm_reduce(PvmMax, &myvals, 10,  PVM_INT,
                                msgtag, "workers",  roottid);
```

포트란:

```
CALL PVMFREDUCE(PvmMax, MYVALS, COUNT,
                INT4, MTAG, 'workers',
                ROOT, NFO)
```

오유:

이름	가능한 경우
PvmBadParam	무효한 변수
PvmNoInst	호출한 과제가 그룹에 없다.
PvmSysErr	국부pvmd가 응답하지 않는다.

pvm_reg_hoster()

새로운 병렬가상기계처리기들을 첨가하기 위하여 이 과제를 등록한다.

문법

C:

```
#include <pvmsdpro.h>
int info = pvm_reg_hoster()
```

변수

info – 귀환상태코드

설명

pvm_reg_hoster는 호출한 과제를 병렬가상기계의 종속 pvmd시동자로 등록한다. 주pvmd는 DM_ADD통보를 접수하면 새로운 종속 pvmd프로쎄스를 생성하는것이 아니라 이 통보를 hoster에게 넘긴다.

주 pvmd가 DM_ADD통보를 받으면(가상기계에 처리기를 첨부하라는 요구)새로운 처리기의 IP주소를 얻고 만일 그것이 생성되였으면 처리기로부터 변수를 얻고 그것을 기정의 변수로 설정한다. 다음에 프로쎄스를 생성하든가 hoster가 등록되여있으면 그에게 SM_STHOST통보를 보낸다. SM STHOST통보의 형식은 다음과 같다.

```
int nhosts          // 호스트들의 수
```

int tid // 호스트의 tid
string options // 호스트파일 so=마당으로부터
string login // 형식 “[username@]hostname.domain” 에서
string command // 원격호스트에서 실행하기 위하여

$PVM_ROOT/lib/pvmd -s -d8 -nhonk 1 80a9ca95:0f5a 4096 3 80a95c43:0000

그리고 종속 pvmd는 다음과 같이 응답한다.

ddpro<2312> arch<ALPHA> ip<80a95c43:0b3f> mtu<4096>

완료하면 hoster는 SM_STHOSTACK통보를 송신자의 주소(주 pvmd)에 돌려보낸다.
응답통보의 형식은 다음과 같다.

int tid ;
string status;

응답하는 TID들은 요구한것들과 일치해야 한다. 그러나 순서는 차이날수 있다.
hoster는 기정으로 PvmDSysErr 혹은 PvmCantStart를 돌려주며 종속 pvmd 그 자체는 PvmDupHost를 돌려준다.
hoster과제는 pvm_setopt(PvmResvTids, 1)를 리용하여 예약된 통보들을 보낼수 있다. 통보들은 자료형식 PvmDataFoo를 리용하여 압축해야 한다.

pvm_reg_rm()

병렬가상기계의 자원관리자로 이 과제를 등록한다.

문법

C:
```
#include <pvmsdpro.h>
int info = pvm_reg_rm( struct hostinfo **hip )
struct hostinfo
 {
    int hi_tid;
    char *hi_name;
    char *hi_arch;
    int hi_speed;
  } hip;
```
변수

hostp – pvmd의 과제id, 이름, 구성방식, 관련속도와 같은 매 처리기관련정보들을 가지고있는 구조체배렬에로의 지적자

info – 상태코드, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_reg_rm()은 호출한 과제를 병렬가상기계과제로 등록하며 종속처리기일정작성기로 등록한다. 이것은 계획화전략에 대응하기 위하여 특정의 libpvm이 다른 과제들과 간섭한다는것을 의미한다.

계획화기구는 체계의 실패와 같은 통지들을 pvmd들에 보내는 봉사요구들을 가지고있는 통보들을 과제들로부터 접수한다.

주의:

이것은 작은 과제가 아니다. 이것은 단순한 순환식과제분배방법으로는 호출할수 없으며 특수한 일정작성규칙에 따라야 한다.

이러한 통보들의 의미를 리해하자면 병렬가상기계의 원천코드를 해석하거나 지도서를 참고하여야 한다.

Libpvm 호출	계획화통보	일반 통보
pvm_addhosts()	SM_ADDHOST	TM_ADDHOST
pvm_config()	SM_CONFIG	TM_CONFIG
pvm_delhosts()	SM_DELHOST	TM_DELHOST

pvm_notify()	SM_NOTIFY	TM_NOTIFY
pvm_spawn()	SM_SPAWN	TM_SPAWN
pvm_tasks()	SM_TASK	TM_TASK
pvm_reg_sched()	SM_SCHED	TM_SCHED

자원관리자들은 차례로 다음의 통보들을 구성하여 pvmd들에 전송하여야 한다.

계획화통보	표준통보
SM_EXEC	DM_EXEC
SM_EXECACK	DM_EXECACK
SM_ADD	DM_ADD
SM_ADDACK	DM_ADDACK
SM_HANDOFF	(none)

아래의 통보들은 체계가 비동기적으로 자원관리자에 보내준다.

계획화통보	의미
SM_TASKX	과제의 완료/실패를 통지한다.
SM_HOSTX	처리기가 삭제/실패하였음을 통지한다.

자원관리자는 pvm_setopt(PvmResvTids, 1)를 리용하여 예약된 통보를 보내게 하여야 한다.

통보들은 자료형식 PvmDataFoo를 리용하여 압축한다.

pvm_reg_tasker()

새로운 병렬가상기계과제들을 생성하기 위하여 이 과제를 등록한다.

문법

C:

```
#include <pvmsdpro.h>
int info = pvm_reg_tasker()
```

변수

info – 상태코드

설명

pvm_reg_tasker는 호출한 과제를 병렬가상기계의 과제시동자로 등록한다. tasker가 pvmd와 함께 등록되고 pvmd가 DM_EXEC통보를 접수하면 fork()나 exec()로 과제를 실행하지 않고 tasker에 이 통보를 넘겨준다.

pvmd는 DM_EXEC통보를 접수하면(새로운 과제들을 실행하라는 요구)파일이름을 가지고 그 파일이 들어있는 등록부를 조사한다. 그 파일을 찾으면 fork()와 exec()를 리용하여 과제를 실행하며 tasker가 등록되여있는 경우에는 SM_STTASK를 그에게 보낸다.

SM_STTASK통보의 형식은 다음과 같다.

```
int tid;
int flags;
string path;
int argc;
string argv[argc];
int nenv;
string env[nenv];
```

tasker는 이 통보를 받으면 프로쎄스를 실행한다. 만일 과제가 성공적으로 실행한다면 tasker는 pvmd에 응답하지 않는다. 과제는 자체로 pvmd에 다시 련결된다. 만일 어떤 과제가 완료하면 tasker는 SM_TASKX통보를 pvmd에 보내야 한다. SM_TASKX통보의 형식은 다음과 같다.

```
int tid;
int status;
```

int u_sec;
int u_usec;
int s_sec;
int s_usec;

tasker는 pvm_setopt(PvmResvTids, 1)를 리용하여 예약된 통보들을 보내야 한다. 통보들은 PvmDataFoo형식으로 압축하여야 한다.

pvmfsend() pvm_send()

능동통보완충기에 자료를 보낸다.

문법

C:

```
int info = pvm_send( int tid, int msgtag )
```

포트란:

```
call pvmfsend( tid, msgtag, info )
```

변수

tid – 목적프로쎄스의 과제식별자

msgtag – 리용자가 지적한 통보표적, msgtag >= 0

info – 상태코드, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_send는 능동송신완충기에 기억된 통보를 tid로 지적한 병렬가상기계프로쎄스에 보낸다. msgtag는 통보의 내용을 표시하기 위하여 리용한다. pvm_send는 성공하면 info = 0 오유가 생기면 info < 0으로 된다.

pvm_send는 비동기적이다. pvm_send는 먼저 목적지가 같은 프로쎄스가 아닌가를 검사한다.

목적지가 같은 처리기이고 이 처리기가 다중처리기이면 제작자가 지원하는 통보전달기능들을 리용하여 프로쎄스들사이에 자료를 이동한다.

실례프로그람

C:

```
info = pvm_initsend( PvmDataDefault );
info = pvm_pkint( array, 10, 1 );
msgtag = 3 ;
info = pvm_send( tid, msgtag );
```
포트란:
```
CALL PVMFINITSEND(PVMRAW, INFO)
CALL PVMFPACK( REAL8, DATA, 100, 1, INFO )
CALL PVMFSEND( TID, 3, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	무효한 tid 혹은 msgtag < 0
PvmSysErr	pvmd가 응답하지 않는다.
PvmNoBuf	능동인 송신완충기가 없다. 송신하기전에 pvm_initsend()를 호출하시오.

pvmfsendsig() pvm_sendsig()

다른 병렬가상기계프로쎄스에 신호기를 보낸다.

문법

C:
```
int info = pvm_sendsig( int tid, int signum )
```
포트란:
```
call pvmfsendsig( tid, signum, info )
```
변수

tid – 신호기를 접수할 병렬가상기계프로쎄스의 과제식별자

signum – 신호기번호

info – 상태코드, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_sendsig는 신호기번호 signum을 tid로 지적한 병렬가상기계프로쎄스에 보낸다. pvm_sendsig가 성공하면 info = 0 오유가 생기면 info < 0으로 된다.

pvm_sendsig는 신호기조종경험을 가지고있는 리용자가 리용하여야 한다. 이 함수는 병렬처리환경에서 예측할수 없는 거동, 교착상태, 체계실패와 같은 사건들을 처리하는데 리용할수 있다.

실례프로그람

C:

```
tid = pvm_parent();
info = pvm_sendsig( tid, SIGKILL);
```

포트란:

```
CALL PVMFBUFINFO( BUFID, BYTES, TYPE, TID, INFO )
CALL PVMFSENDSIG( TID, SIGNUM, INFO )
```

오유:

이름	가능한 경우
PvmSysErr	pvmd가 응답하지 않는다.
PvmBadParam	무효한 tid값

pvmfsetopt()　　　　pvm_setopt()

여러가지 병렬가상기계서고 추가선택들을 설정한다.

문법

C:

```
int oldval = pvm_setopt( int what, int val )
```

포트란:

```
call pvmfsetrbuf( what, val, oldval )
```

변수

what – 무엇을 설정하는가를 지정하는 옹근수, 추가선택들은 다음과 같다.

추가선택	값	의미
PvmRoute	1	경로규칙
PvmDebugMask	2	오유수정 마스크
PvmAutoErr	3	자동오유보고
PvmOutputTid	4	자식과제의 표준출구
PvmOutputCode	5	출구통보표적
PvmTraceTid	6	자식과제들의 추적수단
PvmTraceCode	7	추적통보표적
PvmFragSize	8	통보의 토막크기
PvmResvTids	9	통보들이 예약된 표적들과 TID들에 허용

val – 추가선택값을 지정하는 옹근수, 미리 정의된 값들은 아래와 같다.

추가선택	값	의미
PvmDontRoute	1	련결요구하지 않거나 허가하지 않음
PvmAllowDirect	2	련결요구하지 않고 허가함
PvmRouteDirect	3	련결요구하고 허가함

oldval – 이전에 설정된 추가선택을 돌려주는 옹근수

설명

pvm_setopt는 리용자가 이 함수를 리용하여 병렬가상기계체계에 추가선택들을 설정할수 있게 하는 일반목적함수이다.

PVM 3.2에서 pvm_setopt는 자동오유인쇄, 오유수정준위, 통신경로지정방법들을 포함하여 여러가지 추가선택들을 설정하는데 리용할수 있다. pvm_setopt는 oldval에 이미 설정하였던 추가선택을 돌려준다.

pvmRoute: 통신경로지정인 경우에 pvm_setopt는 직접 과제-과제를 련결하는 PvmRouteDirect를 설정할것을 병렬가상기계에

권고한다. 일단 련결이 수립되면 응용프로그람이 완료할 때까지 유지한다.

만일 두 과제들중 하나가 PvmDontRoute련결을 요구하기때문에 혹은 요구하는 자원이 없어 직접 련결할수 없다면 병렬가상기계데몬을 통한 기정의 경로지정을 리용한다.

인텔파라곤과 같은 다중처리기체계에서는 이 추가선택을 무시한다. 그것은 이 기계들에서는 과제들사이의 통신이 처리기전용의 직접통신 방법을 리용하기때문이다.

pvm_setopt는 여러번 호출하여 교대적으로 직접련결을 실현할수 있다.

pvmAllowDirect는 기정으로 설정한 경로지정추가선택이다.

두 과제들사이에 일단 직접련결을 하면 과제들은 통보를 전송하는데 이것을 리용할수 있다.

PvmDebugMask: 이 추가선택에 대하여 val은 오유수정준위로 된다. 오유수정이 설정되면 병렬가상기계는 동작과정전반에 대한 정보를 구체적으로 기록한다.

PvmAutoErr: 자동오유보고인 경우 오유를 내보내는 임의의 병렬가상 기계함수들은 자동적으로 오유통보를 인쇄한다.

PvmOutputTid: 이 추가선택에 대하여 val은 자식과제의 표준출구장치로 된다. 호출한 과제와 그가 실행시킨 임의의 다른 과제들의 표준출구는 지적한 장치로 출구한다. val은 병렬가상기계의 과제 혹은 pvmd의 tid이다. 0으로 기정설정하면 주처리기의 표준출구로 되며 /tmp/pvml.<uid>에 기록된다.

PvmOutputCode: 이것은 PvmOutputTid를 자기자체에 설정한 경우에 의미가 있다. 이것은 다른 과제들로부터의 표준출구를 포함하고있는 수신통보들에서 리용하는 통보표적이다.

PvmTraceTid: 이 추가선택에 대하여 val은 호출한 프로쎄스와 그의 자식과제들에 대한 추적사건들을 기록하는 과제를 지적한다. val은

병렬가상기계과제 혹은 pvmd의 과제id이다. val을 0으로 기정설정하면 주처리기에로 추적사건을 돌린다.

PvmTraceCode: 이것은 PvmTraceTid를 자기자체에 설정한 경우에 의미가 있다. 이것은 다른 과제들로부터의 추적표준출구를 포함하고있는 수신통보들에서 리용하는 통보표적이다.

PvmFragSize: 이 추가선택에 대하여 val은 바이트로 된 통보의 토막크기를 지적한다.

실례프로그람

C:

```
oldval = pvm_setopt( PvmRoute, PvmRouteDirect );
```

포트란:

```
CALL PVMFSETOPT( PVMAUTOERR, 1, OLD VAL )
```

오유:

이름	가능한 경우
PvmBadParam	무효한 변수

pvmfsetrbuf() pvm_setrbuf()

능동수신완충기를 절환하고 그것을 이전의 완충기에 보관한다.

문법

C:

```
int oldbuf = pvm_setrbuf( int bufid )
```

포트란:

```
call pvmfsetrbuf( bufid, oldbuf )
```

변수

bufid – 새로운 능동수신완충기에 대한 통보완충기식별자를 지적하는 옹근수

oldbuf – 이전의 능동수신완충기에 대한 통보완충기식별자를 돌려주는 옹근수

설명

pvm_setrbuf는 능동수신완충기를 bufid로 절환하고 그것을 이전의 완충기 oldbuf에 보관한다. bufid = 0으로 설정하면 현재의 능동수신완충기가 보관되며 능동인 수신완충기는 존재하지 않는다. 수신이 성과적으로 진행되면 새로운 능동수신완충기가 창조된다.

만일 이전에 수신한 통보를 풀지 않고 보관할것을 요구한다면 이전의 bufid를 보관하고 후에 풀기 위해 능동수신완충기를 재설정할수 있다. 이 함수는 다중통보완충기들을 관리할 때 필요하다.

실례프로그람

C:

```
rbuf1 = pvm_setrbuf( rbuf2 );
```

포트란:

```
CALL PVMFSETRBUF( NEWBUF, OLDBUF )
```

오유:

이름	가능한 경우
PvmBadParam	무효한 bufid
PvmNoSuchBuf	존재하지 않는 통보완충기로 절환하였다.

pvmfsetsbuf() pvm_setsbuf()

능동송신완충기에로 절환한다.

문법

C:

```
int oldbuf = pvm_setsbuf( int bufid )
```

포트란:

```
call pvmfsetsbuf( bufid, oldbuf )
```

변수

bufid – 새로운 능동송신완충기를 위한 통보완충기식별자, 기정값은 0으로 설정한다.

oldbuf – 이전의 능동송신완충기를 위한 통보완충기식별자

설명

pvm_setsbuf는 능동송신완충기를 bufid로 절환하고 그것을 이전의 완충기 oldbuf에 보관한다.

bufid = 0으로 설정하면 현재의 능동송신완충기가 보관되며 능동인 송신완충기는 존재하지 않는다.

실례프로그람

C:

```
sbuf1 = pvm_setsbuf( sbuf2 );
```

포트란:

```
CALL PVMFSETSBUF( NEWBUF, OLDBUF )
```

오유:

이름	가능한 경우
PvmBadParam	무효한 bufid
PvmNoSuchBuf	존재하지 않는 통보완충기

pvmfspawn() pvm_spawn()

새로운 병렬가상기계과제를 시동한다.

문법

C:

```
int numt = pvm_spawn( char *task, char **argv,
           int flag, char *where,int ntask, int *tids )
```

포트란:

```
call pvmfspawn( task, flag, where, ntask, tids, numt )
```

변수

task – 실행가능한 이름을 가지고있는 문자렬, 실행프로그람은 이미전에 실행하는 처리기에 있어야 한다. 기정의 위치는 $HOME/pvm3/bin/$PVM_ARCH/filename 이다

argv – 배렬의 끝이 null문자로 끝나는 실행프로그람의 변수배렬에로의 지적자, 만일 프로그람이 변수를 가지지 않으면 이 명령의 두번째 변수는 null로 된다.

flag – 실행추가선택을 지적하는 옹근수, C에서 flag는 다음 추가선택들의 합으로 된다.

추가선택	값	의미
PvmTaskDefault	0	PVM은 과제를 실행할 임의의 호스트를 선택할수 있다.
PvmTaskHost	1	구체적인 처리기를 지적한다.
PvmTaskArch	2	구성방식을 지정한다.
PvmTaskDebug	4	오유수정도구우에서 프로쎄스를 실행한다.
PvmTaskTrace	8	프로쎄스는 PVM추적자료를 생성한다.
PvmMppFront	16	MPP의 front-end마디에서 프로쎄스를 실행한다.
PvmHostCompl	32	처리기의 보모임(지적된 처리기가 속하지 않은 남은 처리기들)을 리용한다.

where – 프로쎄스를 어디서 실행시키는가를 지적하는 문자렬 flag의 값에 따라 처리기의 이름은 “ibm1.epm.ornl.gov" 혹은 “SUN4”와 같은 PVM의 구성방식일수 있다. flag = 0이면 where는 무시되며 병렬가상기계는 가장 적절한 처리기를 선택한다.

ntask – 실행할 프로그람개수를 지적하는 옹근수

tids – 적어도 ntask길이를 가진 배렬, 귀환하면 배렬은 pvm_spawn호출에 의해 시동된 병렬가상기계프로쎄스들의 tid들을 가지고있다. 만일 어떤 과제를 실행할 때 오유가 생기면 배렬에서 그 과제에 해당한 위치에는 오유코드가 설정된다.

numt – 실행되는 과제들의 실제개수가 설정된다. 오유가 생기면 0보다 작은 값이 설정된다. 부분적인 오유가 생기면 ntask보다 작은 값이 설정된다.

설명

pvm_spawn은 task라는 이름을 가진 ntask개의 프로그람들을 실행한다. 환경변수를 제공하는 체계에서는 이미 환경에 설정된 환경변수가 아들과제들에 넘어간다. 즉 환경변수 PVM_EXPORT가 넘어간다.

PVM_EXPORT가 “:”으로 다른 변수를 구분하면 그것들도 넘어간다. 이것은 실례로 다음과 같은 경우 쓸모있다.

```
setenv DISPLAY myworkstation:0.0
setenv MYSTERYVAR 13
setenv PVM_EXPORT DISPLAY:MYSTERYVAR
```

병렬가상기계프로쎄스들을 실행하는 처리기들은 flag와 where변수로 지적한다. 귀환하면 tids배렬은 실행된 매개 과제들의 식별자들을 가지고있다.

pvm_spawn는 한개 이상의 과제들을 실행할수 있는데 numt는 실행할 과제들의 실지개수를 가지고있다.

체계오유가 발생하면 numt < 0으로 된다. 만일 numt =< ntask이면 일부 프로그람들이 실행시 오유가 발생하였다는것을 보여주며 이때 리용자는 관련한 오유코드들을 가지고있는 tids배렬에서 오유원인들을 조사할수 있다. 배렬에서 첫 numt개의 tids들은 정상이며 따라서 이 과제들과는 정상적인 통신을 실행할수 있다.

flag가 0으로 설정되고 where가 NULL (혹은 포트란에서는 "*")로 설정되면 계발식방법을 적용하여 ntask개의 프로쎄스들을 가상기계에 분배한다. 현재 계발식방법은 표에 등록된 처리기들에 차례로 과제들을 분배하는 균등분배이다.

만일 PvmHostCompl기발이 설정되면 결과의 처리기모임은 지적된 처리기가 포함되지 않은 보조모임으로 된다. 또한 TaskHost로 지적한 처리기이름은 국부처리기로서 주어진다.

where에 다중처리기가 설정되면 pvm_spawn은 제작자전용의 기능을 리용하여 ntask개의 과제들을 이 단일처리기우에서 실행한다.

PvmTaskDebug가 설정되면 pvmd는 오유수정도구우에서 과제들을 실행한다.

이 경우 pvm3/bin/ARCH/등록부에 있는 과제들을 실행하는것이 아니라 pvm3/lib/debugger를 실행한다.

오유제거기는 리용자들이 개별적인 과제들을 수정할수 있는 쉘각본 프로그람이다.

실례프로그람

C:

```
numt = pvm_spawn( "host", 0, PvmTaskHost,
                  "sparky", 1, &tid[0] );
numt = pvm_spawn( "host", 0, (PvmTaskHost+
       PvmTaskDebug), "sparky", 1, &tid[0] );
numt = pvm_spawn( "node", 0, PvmTaskArch,
                  "RIOS", 1, &tid[i] );
numt = pvm_spawn( "FEM1", args, 0, 0, 16,
                   tids );
numt = pvm_spawn( "pde", 0, PvmTaskHost,
                 "paragon.ornl", 512, tids );
```

포트란:

```
FLAG = PVMARCH + PVMDEBUG
CALL PVMFSPAWN( 'node', FLAG, 'SUN4', 1,
                TID(3), NUMT )
CALL PVMFSPAWN( 'FEM1', PVMDEFAULT, '*',
                 16, TIDS, NUMT )
CALL PVMFSPAWN( 'TBMD', PVMHOST,
           'cm5.utk.edu', 32, TIDS, NUMT )
```

오유:

이름	값	가능한 경우

PvmBadParam -2 무효한 변수가 지정
PvmNoHost -6 지적한 처리기가 가상기계에 없다.
PvmNoFile -7 지적한 프로그람을 찾을수 없다.
기정위치는 ~/pvm3/bin/$ARCH이다.
여기서 ARCH는 PVM의 구성방식이름
이다.
PvmNoMem -10 malloc가 실패하였다. 기억기부족
PvmSysErr -14 pvmd가 응답하지 않는다.
PvmOutOfRes -27 자원이 부족

pvmftasks() pvm_tasks()

가상기계에서 실행하는 과제들에 대한 정보를 준다.

문법
C:

```
      int info = pvm_tasks( int where, int *ntask,
                              struct pvmtaskinfo **taskp )
struct pvmtaskinfo
   {
      int ti_tid;
      int ti_ptid;
      int ti_host;
      int ti_flag;
      char *ti_a_out;
       int ti_pid;
    } taskp;
```

포트란:

```
     call pvmftasks( where, ntask, tid, ptid,
                     dtid, flag, aout, info )
```

변수
where – 정보를 요구하는 과제를 지적한다, 추가선택들은 다음과
갈다.

0	가상기계에 있는 모든 과제들
pvmd tid	주어진 처리기의 모든 과제들
tid	지적한 과제

ntask – 실행하는 과제들의 개수를 지적
taskp – 그의 과제id, 어미과제의 tid, pvmd의 과제 id, 상태기발, 이 과제의 프로그람이름, 과제의 프로쎄스 id (OS에 의존)와 같은 정보들을 가지고있는 구조체배렬에로의 지적자
tid – 어떤 과제의 과제ID를 지적
ptid – 어미과제의 과제id를 지적
dtid - 과제가 있는 처리기의 pvmd과제의 과제 id
flag – 과제의 상태
aout – 시동한 과제의 과제이름, 수동적으로 시동한 과제는 공백을 돌려준다.
info – 이 함수가 돌리는 상태코드, 오유가 생기면 0보다 작은 값이 설정

설명

pvm_tasks는 가상기계에서 실행하는 과제들에 대한 정보를 준다. 이 정보는 조작탁지령 ps로 얻는 정보와 같다.

C함수는 한번의 호출로 가상기계전체에 대한 정보를 얻는다.

포트란함수는 한번의 호출로 하나의 과제에 대한 정보를 얻으므로 가상기계 전체에 대한 과제정보를 얻으려면 과제개수만큼 호출을 반복하여야 한다.

pvm_tasks는 성공하면 info = 0으로 되며 오유가 생기면 info < 0으로 된다.

실례프로그람

C:

```
info = pvm_tasks( 0, &ntask, &taskp );
```

포트란:

```
CALL PVMFTASKS( DTID, NTASK, INFO )
```

오유:

이름	가능한 경우
PvmBadParam	where변수가 틀렸다.
PvmSysErr	pvmd가 응답하지 않는다.
PvmNoHost	가상기계에는 그런 처리기가 없다.

pvmftidtohost() pvm_tidtohost()

지적된 과제가 실행되는 마디점콤퓨터의 처리기번호를 되돌린다.

문법

C:

```
int dtid = pvm_tidtohost( int tid )
```

포트란:

```
call pvmftidtohost( tid, dtid )
```

변수

tid – 문의하는 PVM 프로쎄스의 옹근수과제식별자

dtid – 마디점콤퓨터에서 실행하는 데몬의 과제식별자를 되돌리는 옹근수, 혹은 오유가 있으면 부의 옹근수

설명

이 함수는 tid로 식별되는 프로쎄스가 있는 마디점콤퓨터 식별자를 돌려준다.

실례프로그람

C:

```
host = pvm_tidtohost( tid[0] );
```

포트란:

```
CALL PVMFTIDTOHOST(TID, HOSTID)
```

오유:

이름	가능한 경우
pvmBadParam –	무효인 tid값이 지적되였다.

PvmSysErr — pvmd가 응답없다.

pvmftrecv() pvm_trecv()

기다림한계시간을 가지고 수신한다.

문법

C:

```
int bufid = pvm_trecv( int tid, int msgtag,
                       struct timeval *tmout )
```

포트란:

```
call pvmftrecv( tid, msgtag, sec, usec, bufid )
```

변수

tid – 송신프로세스의 과제식별자

msgtag – 통보표적, 이것은 0보다 작지 말아야 한다.

tmout – 통보를 받지 못하였을 때 귀환하기전에 기다리는 시간

sec, usec – 기다리는 시간지정값

bufid – 새로운 능동수신완충기의 값, 오유가 생기면 0보다 작은 값이 설정된다.

설명

pvm_trecv는 msgtag라는 표식을 단 통보가 tid로부터 도착할 때까지 프로세스를 봉쇄한다. pvm_trecv는 다음 통보를 새로운 능동수신완충기에 배치하며 현재 수신완충기를 지운다.

지적한 기다림 시간내에 일치하는 통보가 도착하지 않으면 pvm_trecv는 통보가 없이 귀환한다. msgtag = -1이든가 tid = -1이면 임의의 통보를 접수한다. 이때 리용자는 아래의 추가선택들중 임의의 추가선택을 리용할수 있다.

tid = -1 이고 msgtag를 리용자가 정의하였다면 pvm_trecv는 임의의 프로세스로부터 msgtag를 가진 통보를 접수할수 있다.

msgtag = -1이고 tid 를 리용자가 정의하였다면 pvm_trecv는 프로세스 tid가 보내는 임의의 통보를 접수한다.

tid = -1 이고 msgtag = -1이면 pvm_trecv는 임의의 프로세스로부터 임의의 통보를 접수한다.

C에서 시간마당 tv_sec와 tv_usec는 pvm_trecv가 얼마나 오래동안 통보를 기다려야 하는가를 지적한다.

두 값을 다 0으로 설정하면 pvm_trecv는 pvm_nrecv()처럼 동작하는데 이것은 통보를 조사하고 일치하는것이 없으면 즉시 돌아온다.

C에서 이 마당에 null지적자를 넘기면 pvm_trecv는 pvm_recv()처럼 동작한다. 즉 무한대기한다.

포트란에서는 sec를 0으로 설정하면 같은 효과를 가진다.

병렬가상기계모델은 통보의 순서를 보존한다.

만일 과제 1이 통보 A를 과제 2에 보내고 그 다음 과제 1이 통보 B를 과제 2 에 보내면 통보 A는 통보 B보다 과제 2에 먼저 도착한다. 더우기 두 통보들이 과제 2가 접수하기전에 도착해도 항상 통보 A를 먼저 접수한다.

pvm_trecv는 성공하면 능동수신완충기의 식별자가 bufid에 설정된다. 오유가 생기면 bufid < 0으로 된다.

일단 pvm_trecv가 귀환하면 통보에 있는 자료는 풀기함수를 리용하여 리용자의 기억기에 풀린다.

실례프로그람

C:

```
struct timeval tmout;
tid = pvm_parent();
msgtag = 4 ;
if ((bufid = pvm_trecv( tid, msgtag, &tmout )) >0) {
    pvm_upkint( tid_array, 10, 1 );
    pvm_upkint( problem_size, 1, 1 );
    pvm_upkfloat( input_array, 100, 1 );
 }
```

포트란:

```
CALL PVMFRECV( -1, 4, 60, 0, BUFID )
IF (BUFID .GT. 0) THEN
CALL PVMFUNPACK( INTEGER4, TIDS, 25, 1, INFO )
```

CALL PVMFUNPACK(REAL8, MATRIX, 100, 100, INFO)
ENDIF

오유:

이름	가능한 경우
PvmBadParam	무효 tid값 혹은 msgtag <-1
PvmSysErr	pvmd가 응답하지 않는다.

pvmfunpack()　　　　　　pvm_upk*()

능동통보완충기를 기입된 자료형을 가진 배렬로 푼다.

문법

C:

```
int info = pvm_unpackf( const char *fmt, ... )
int info = pvm_upkbyte( char *xp, int nitem, int stride )
int info = pvm_upkcplx( float *cp, int nitem, int stride )
int info = pvm_upkdcplx( double *zp, int nitem, int stride )
int info = pvm_upkdouble(double *dp, int nitem, int stride )
int info = pvm_upkfloat( float *fp, int nitem, int stride )
int info = pvm_upkint( int *ip, int nitem, int stride )
int info = pvm_upklong( long *ip, int nitem, int stride )
int info = pvm_upkshort( short *jp, int nitem, int stride )
int info = pvm_upkstr( char *sp )
```

포트란:

```
call pvmfunpack( what, xp, nitem, stride, info )
```

변수

fmt – 무엇을 압축하는가를 지정하는 인쇄가능한 형식의 표기

nitem – 풀어야 할 항목들의 총 개수

stride – 항목들을 압축할 때 리용한 걸음, 실례로 pvm_upkcplx에서 stride= 2이면 다른 복소수를 푼다.

xp – 바이트블로크의 시작점에로의 지적자, 임의의 자료형이 될수 있지만 대응하는 압축자료형과 일치해야 한다.

cp – 적어도 nitem*stride 개의 항목들을 가지고있는 복소수배렬

zp – 적어도 nitem*stride개의 항목들을 가지고있는 배정확도 복소수배렬

dp – 적어도 nitem*stride개의 항목들을 가지고있는 배정확도 실수배렬

fp – 적어도 nitem*stride개의 항목들을 가지고있는 실수배렬

ip – 적어도 nitem*stride개의 항목들을 가지고있는 옹근수배렬

jp – 적어도 nitem*stride개의 항목들을 가지고있는 옹근수*2배렬

sp – null로 끝나는 문자렬

what – 풀기할 자료의 형을 지적하는 옹근수

	what	추가선택	
STRING	0	REAL4	4
BYTE1	1	COMPLEX8	5
INTEGER2	2	REAL8	6
INTEGER4	3	COMPLEX16	7

info – 상태코드, 오유가 생기면 0보다 작은 값이 설정한다.

설명

매 pvm_upk* 함수들은 능동수신완충기로부터 주어진 자료형의 배렬들을 푼다. 매 함수에서 변수는 풀기할 배렬에로의 지적자이며 nitem은 풀 항목들의 전체 개수, stride는 풀기시에 리용하는 걸음을 지적한다.

여기서 례외는 pvm_upkstr()인데 이것은 정의에 따라 null로 끝나는 문자렬을 풀며 따라서 nitem, stride와 같은 변수를 요구하지 않는다.

포트란함수 pvmfunpack(STRING, ...)에서는 문자렬에 있는 문자들의 개수로 nitem을 취급하며 stride =1로 한다.

이 함수는 성공하면 info = 0이며 오유가 생기면 info < 0으로 된다.

단일 변수는 nitem=1 , stride=1 로 하여 풀수 있다.

pvm_unpackf()는 인쇄가능한 형식의 표기를 리용하여 자료를 무엇으로 또 어떻게 압축하여 송신완충기에 보관하는가를 지적한다. 모든 변수는 주소로서 넘어간다. BNF와 류사한 형식적서술법이 리용된다.

format : null | init | format fmt
init : null | '%' '+'
fmt : '%' count stride modifiers fchar
fchar : 'c' | 'd' | 'f' | 'x' | 's'
count : null | [0-9]+ | '*'
stride : null | '.' ([0-9]+ | '*')
modifiers : null | modifiers mchar
mchar : 'h' | 'l' | 'u'

형식:
+ 는 initsend를 의미하며 param목록에 있는 int와 일치해야 한다.
c 압축/풀기할 바이트를 지적한다.
d 옹근수
f 류점수
x 배정도 류점수
s 문자렬

략자:
h short (int)
l long (int, float, complex float)
u unsigned (int)
'*' count 나 stride는 param목록에 있는 int와 일치해야 한다.

실례
C:
info = pvm_recv(tid, msgtag);
info = pvm_upkstr(string);

```
info = pvm_upkint( &size, 1, 1 );
info = pvm_upkint( array, size, 1 );
info = pvm_upkdouble( matrix, size*size, 1 );
```
포트란:
```
CALL PVMFRECV( TID, MSGTAG );
CALL PVMFUNPACK( INTEGER4, NSIZE, 1, 1, INFO )
CALL PVMFUNPACK( STRING, STEPNAME, 8, 1,
                 INFO )
CALL PVMFUNPACK( REAL4, A(5,1), NSIZE,
                 NSIZE ,INFO )
```
오유:

이름	가능한 경우
PvmNoData	수신완충기의 끝을 초과하여 읽었다. 이 오유는 대체로 완충기에 본래 압축한 개수보다 더 많은 항목들을 풀려고 할 때 나타난다.
PvmBadMsg	수신한 통보를 해신할수 없다. 이 오유는 대체로 처리기들이 서로 다른 기종일 때 나타난다. 이때는 PvmDataDefault로 해신해보시오.
PvmNoBuf	풀기할 능동인 수신완충기가 없다.